CONTABILIDAD EN LA NUBE

Carlos Martínez Sacristán
Enrique Ugedo Egido

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA DE COMPUTADORES Y
GRADO EN INGENIERÍA DEL SOFTWARE
Curso académico 2016/2017

Madrid, 9 de febrero de 2017

Director: Luis Fernando Llana Díaz
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Índice

Índice de figuras

Resumen

Partiendo de la aplicación de contabilidad GNUCash, que proporciona un entorno en el que llevar las cuentas de gastos e ingresos, realizamos una aplicación web que utiliza la misma base de datos, pero siendo en este caso multiplataforma.

Las modificaciones llevadas a cabo en esta aplicación serán interpretadas por GNUCash la siguiente vez que utilicemos la aplicación de escritorio para abrir la base de datos.

Sus usuarios ya no necesitarán memorizar o anotar en otro lugar los movimientos monetarios del día a día para anotarlos en GNUCash al llegar a casa, sino que podrán introducirlos directamente en su móvil.

Palabras clave: Contabilidad, finanzas personales, economía, ahorros, cuentas, bancos, crédito.

Abstract

The starting point of our project is the application GNUCash, that is an accounting application. We have started a Web application that uses the same database as GNUCash. So, we can use the application everywhere on any device.

The modifications carried out on the database are compatible with those of GNUCash. So, both programs can be used indistinctly.

The users' application will not need to memorize or to annotate in some other place their daily movements. Instead, they can annotate it directly in the GNUCash database one their mobile.

Key words: Accounting, personal finance, economics, savings, accounts, banks, credit.

Capítulo 1 Introducción

La contabilidad personal y familiar siempre ha sido una gran preocupación. Quien más quien menos, lleva de alguna manera sus cuentas de casa, en una hoja de cálculo o algo similar. La necesidad de controlar los gastos y las finanzas personales se ha acentuado especialmente durante los años de la crisis.

Sin embargo, esta tarea suele ser algo engorrosa, sobre todo si el usuario no es experto en la materia. El creciente mercado de aplicaciones móviles no ha sido ajeno a ello y hoy en día existen multitud de aplicaciones con distintas utilidades orientadas a esta finalidad, aunque pocas hay que podamos considerar completas, obligando al usuario a tener multitud de aplicaciones diferentes para cuestiones que estarían mejor integradas en una sola.

GNUCash es una aplicación que guarda los datos en un XML local, o en una base de datos, local o remota. Está disponible únicamente para equipos de escritorio con sistema operativo Linux, macOS o Windows. No existe una aplicación móvil para gestionar esa base de datos, y es precisamente lo que se desarrolla en este proyecto. Se creará una aplicación web, y por ende multiplataforma, que utilice la base de datos en la nube de la que se alimenta GNUCash. La web permitirá mostrar, editar y añadir elementos desde cualquier dispositivo con navegador de internet, sin necesidad de tener la aplicación instalada. Incluso a varios dispositivos conectados simultáneamente.

La utilidad de guardar los datos en una base de datos en lugar de un XML ha sido añadida recientemente a GNUCash, y es lo que ha motivado el planteamiento de este proyecto, al ser ahora posible tener los datos almacenados en un sistema en la nube que esté permanentemente sincronizado. Existían alternativas almacenando el XML mediante servicios de sincronización de archivos, pero esto seguía manteniendo distintas copias locales que podrían no estar siempre sincronizadas. Además, generaría problemas al utilizarse la aplicación simultáneamente en distintos dispositivos, ya que los cambios no se guardan hasta que el usuario cierra la aplicación o pulsa el botón de guardar. Sin embargo, utilizando la base de datos, el programa la consulta cada vez que se actualiza la vista y en cuanto se añade o modifica un movimiento, este es introducido directamente en la base de datos.

La aplicación GNUCash es además muy completa, permitiendo generar informes y gráficas a partir de los datos. Esto, unido a que está desarrollada bajo licencia GNU y el código es abierto, animó a tomar la decisión crear la aplicación web de este proyecto basada en GNUCash y compatible con ella. Eso sí, no se desarrollarán todas las características de GNUCash, pero sí las más básicas. La versión de escritorio de GNUCash constituye por tanto un complemento a la aplicación web de este proyecto ya que permite explotar más los datos que en ella almacenamos.

Con la aplicación que se desarrolla en este proyecto se puede consultar el listado y balance de las cuentas, sus transacciones y añadir nuevos movimientos. Esto producirá los cambios pertinentes en la base de datos para que al abrir en GNUCash desde la aplicación de escritorio, los cambios que el usuario haya modificado se vean reflejados de forma consistente. El principal objetivo es convertir en multiplataforma las operaciones más habituales y cotidianas de GNUCash. Que se puedan realizar de forma rápida y sencilla, desde cualquier lugar.

Para empezar con este desarrollo, al basarse en una aplicación ya existente, ha sido necesario, antes de nada, familiarizarse con esa aplicación, conocer al detalle su funcionamiento interno y el comportamiento de la base de datos al realizar modificaciones. Estudiar la documentación tanto para usuarios como para desarrolladores y comprender la estructura de la base de datos y las relaciones entre las distintas tablas ha sido clave.

Una vez adquirido esos conocimientos, ha sido cuestión de programar de forma que se hiciese compatible la aplicación desarrollada, con todas las pequeñas peculiaridades que posee GNUCash y de las que se hablará en esta memoria.

Chapter 1 Introduction

Household budgeting has always been a great concern for families. Most people do some kind of budget management in the form of a spreadsheet or similar. The need to keep control of the expenditures and personal finance has been increased during the financial crisis.

Nevertheless, this task tends to be cumbersome, even more if we are not experts in the field. The growing market of mobile applications is no foreign to that reality and nowadays there are many applications with different features all pointing to that need but there are few that can be considered complete applications, forcing the user to keep a number of different applications when they should be integrated in one alone.

GNUCash is an application that stores the data in a local XML or a database either local or remote. It is only available for desktop systems with Linux, macOS or Windows operative systems. There is no mobile application for managing GNUCash database and that is precisely what will be developed in this project. We create a web application thus multiplatform that uses the cloud database that is used by GNUCash. We can show, edit and add elements from any device with internet connection without having to install the application. Even multiple devices connected simultaneously.

Being able to store data in a database instead of an XML is a feature that has been recently added to GNUCash and have prompted the approach of this project, being able to have access to cloud stored data that is permanently synchronized. There were alternative approaches storing the XML through file synchronization but this would require to keep local copies that may not be always synchronized. It could also create problems when using the application from different devices simultaneously, giving that changes are not saved until the user closes the application or presses the save button. However, GNUCash queries the database every time a view is updated and on adding or modifying a transaction it is introduced into the database.

GNUCash is also very complete, allowing to generate reports and charts from the data. This, coupled with the fact that is developed under GNU licence and open source encouraged us to create the web application based on and compatible with GNUCash. Our application won't have every feature of GNUCash but the basic ones. The desktop version becomes then a complement to the we app given that it allows to exploit better the data.

The application that is developed in this project can look up the account list and cash balance, check the transactions and add new ones. This will produce the necessary changes in the database so that when opening the desktop version the changes will be reflected in a consistent manner. The main objective is to port to multiplatform the most usual and common operations of GNUCash. Being able to access it in an easy and fast way from anywhere.

As starting point of the development, giving that it will be based on an existing application, it has been mandatory to become familiar with the actual application. Learning the inner working and the database behaviour when making changes. Studying both user and technical documentation for developers and comprehending the structure of the database and relationships between the different tables is key.

Once this knowledge is acquire, making the web application compatible with every little (peculiarity/nuance) that GNUCash possesses has been the objective of the development.

Capítulo 2 Estado del arte

En este capítulo se detallan cada una de las tecnologías utilizadas para desarrollar la aplicación web y el programa de escritorio en el que está basada. Se hace un repaso por otras tecnologías que habrían sido posibles alternativas para desarrollar esta aplicación. Para finalizar el capítulo con un vistazo a aplicaciones similares a la que se presenta en este proyecto.

Sección 2.1 Python

Python es un lenguaje de programación orientado a objetos que se caracteriza por su versatilidad y simplicidad. Es independiente de plataforma y está preparado para crear cualquier tipo de programa. Permite crear aplicaciones web, aplicaciones de escritorio que sean multiplataforma, o scripts en shell para automatizar tareas o procesos de un sistema. En general se puede crear prácticamente cualquier tipo de programa. No es un lenguaje específico de desarrollo web, pero ésta se encuentra entre sus posibilidades.

Por otra parte, es un lenguaje interpretado, lo que significa que el código no debe ser compilado antes de ejecutarse (el intérprete de Python realiza una compilación implícita, de forma que es transparente para el desarrollador), lo que permite que haya una mayor velocidad de desarrollo, pero a su vez tiene la desventaja de una ejecución más lenta que otros lenguajes que necesitan compilarse antes de ejecutarse.

Es un lenguaje por lo general fácil de aprender, ya que ofrece una sintaxis clara y muy visual, basada en la indentación obligada, es decir, se debe tabular el código ya que es el mecanismo que sirve para indicar los bloques del programa, como por ejemplo, un bucle o una función.

Además es un lenguaje que se ha hecho muy popular gracias a la rapidez y sencillez con la que se crean programas y a la gran cantidad de librerías, funciones y tipos de datos incorporados en el propio lenguaje, que permiten reutilizar código y evitar realizar muchas tareas desde cero.

2.1.a) Estructuras de datos

En Python, aparte de los tipos básicos de datos, existen otros tipos más complejos que se denominan colecciones. Las colecciones sirven para agrupar elementos y se dividen en listas, tuplas y diccionarios.

Las listas son colecciones ordenadas, equivalente a lo que en otros lenguajes serían arrays o vectores, pero que pueden contener cualquier tipo de dato, enteros, cadenas, booleanos u otras listas, así como cualquier tipo de objeto o tipo abstracto, como por ejemplo un modelo.

Las tuplas son registros inmutables, que no se pueden modificar después de su creación. Sus elementos tienen definido un orden. Se definen como las listas pero con paréntesis en vez de corchetes. Sus elementos deben estar separados por comas. Al ser inmutables, no tienen métodos que permitan insertar o eliminar elementos, al contrario que las listas, pero tienen la ventaja de ser más rápidas que éstas. Pueden utilizarse como claves en un diccionario, y pueden convertirse en listas (y viceversa).

Los diccionarios son colecciones no ordenadas. Se accede a sus valores mediante una clave, es decir, que en lugar de acceder a los valores mediante un índice, se puede acceder mediante claves que pueden ser de diversos tipos. Las claves son únicas y el acceso al valor asociado es directo.

Sección 2.2 Django

Django es un framework de código abierto con licencia BSD. Escrito en Python y para utilizar con Python, destinado al desarrollo web. Sigue un patrón que se asemeja mucho al patrón MVC (Model-View-Controller), y al que se le llama MVT (Model-View-Template), en el que el controlador sería llamado vista, y la vista plantilla (o *template*). En Django la vista describe qué datos serán presentados y la plantilla describe como se verán. Además genera una interfaz para administrar la base de datos.

Una aplicación desarrollada en Django es autocontenida y permite de facto manejar el acceso a la base de datos y generar sus vistas en HTML así como los formularios necesarios para añadir y modificar entradas.

Django pone énfasis en la reutilización, conectividad y extensibilidad de componentes, respetando el principio DRY (Don't Repeat Yourself, en español No te repitas). Así por ejemplo incorpora un sistema de vistas que

se podrían considerar genéricas, evitando reescribir la lógica de tareas comunes y un sistema de plantillas basado en etiquetas y que soporta herencia.

Las URL se definen mediante expresiones regulares y su visita inicia la ejecución de la vista correspondiente, en la que puede haber cualquier tipo de función desarrollada en Python para atender a las necesidades concretas de cada momento, operación o consulta. Incluye protección CSRF para asegurarse de que los datos introducidos en formularios son válidos y acordes a los tipos definidos en los modelos relacionados con la base de datos.

2.2.a) Formularios web

Una de las principales funciones de Django es la creación de formularios. Incorpora la librería *Forms* que permite crear formularios definidos desde cero o creando un *ModelForm* y se guardará el resultado del formulario en un modelo. Al utilizar *ModelForm*, se indican qué campos de un modelo se quieren incluir en el formulario e incluso es posible personalizar algunas opciones más del mismo. Se pueden utilizar campos de distintos modelos y definir un campo con un valor determinado por una función, cosa que será muy útil en la implementación de este proyecto. Posteriormente Django crea automáticamente el código HTML a partir de ese *ModelForm*. Al recibirse el formulario relleno, desde la vista puede gestionarse al gusto. Estos datos se introducen en el modelo correspondiente y por tanto, como se ha explicado anteriormente, en la base de datos. Es necesario incluir el *token* **CSRF** para evitar inconsistencia en los tipos de datos del formulario y del modelo.

2.2.b) Modelos y bases de datos

Django permite una sincronización perfecta con la base de datos MySQL mediante la creación de modelos (*Models*) en lenguaje Python. Al desarrollar una aplicación en Django, se modificarán los datos recogidos en los modelos, y éstos se replicarán automáticamente en la base de datos MySQL. Existen comandos que crean los modelos a partir de una base de datos ya existente, recuperando también la información del tipo de variable correspondiente a cada una de las filas que conforman la tabla de la BBDD.

Sección 2.3 MySQL

MySQL es un sistema gestor de bases de datos relacionales en SQL. Permite la gestión de una base de datos relacional usando un lenguaje de consulta estructurado. Es una de las mejores soluciones técnicas para gestión de bases de datos y quizás la más utilizada en la actualidad.

Recientes estudios demuestran que MySQL tiene una velocidad superior a la de sus rivales, incluido Oracle, a la hora de ejecutar consultas SQL.

Tiene soporte para bases de datos de grandes dimensiones. Permite hasta 64 índices por tabla.

Es compatible con la práctica totalidad de los Sistemas Operativos existentes. Si a esto se le añade un lenguaje de programación también compatible, como Python, Java o PHP, permite desarrollar aplicaciones Web versátiles y fáciles de migrar a otros SO.

Se ofrece bajo licencia GNU GPL para los usos compatibles. Para incorporarlo a productos privativos es necesario comprar una licencia específica que permita esos usos a la empresa.

Está desarrollado en ANSI C.

2.3.a) phpMyAdmin

Es una herramienta desarrollada en PHP que funciona como editor de bases de datos MySQL y permite también su administración a través de una interfaz web. Con ella se puede ejecutar cualquier sentencia SQL además de hacer operaciones básicas como crear, eliminar y editar bases de datos, tablas y entradas de forma sencilla e intuitiva. Otra opción interesante es la exportación e importación de datos en varios formatos. Se ofrece bajo la licencia GPL.

Sección 2.4 GNUCash

GNUCash es un programa de contabilidad y gestión de finanzas enfocado sobre todo a particulares y pequeñas empresas.

Es un programa de software libre que forma parte del proyecto **GNU** (tiene una licencia GPL) y es multiplataforma (GNU/Linux, Windows, macOS).

Está basado en la doble entrada como sistema de contabilidad, es decir, cada movimiento de una cuenta tiene otra como contrapartida. Por ejemplo, un gasto de 30 euros en un restaurante se anota en la cuenta "Restaurantes", perteneciente a la cuenta "Gastos". Este dinero debe salir de algún sitio, y en este caso (suponiendo que se pague en efectivo), se anotará que salió de la cuenta "Metálico", perteneciente a "Activo".

Es un programa en el que las transacciones se realizan de una forma totalmente transparente y se puede observar cómo se realizan los cálculos y ajustes, al contrario que otros programas de contabilidad más destinados a usuarios finales en los que no podemos ver su funcionamiento interno. Esto, unido a su diseño sencillo y visual, permite aprender rápidamente lo básico sobre contabilidad y cómo funciona el sistema de doble entrada.

GNUCash permite la gestión de distintos tipos de cuentas que se detallarán más adelante. Cuentas de gastos, tarjetas de crédito, préstamos, y prácticamente todo lo que se requiera, ya que es muy configurable tanto para llevar el control de una cuenta real como simplemente para controlar tus gastos.

Al crear un nuevo libro de contabilidad existen distintas opciones predefinidas, entre ellas la de "Cuenta común", que por defecto incluye un árbol de cuentas en el que se engloban la mayoría de gastos o beneficios cotidianos, como sueldo, impuestos, suministros domésticos, comestibles, ocio, gastos médicos, etcétera. A parte se pueden crear otras subcuentas de gastos e ingresos para llevar un control eficiente de éstos, así como programar transacciones en el tiempo y automatizar muchas de estas entradas y salidas de capital (muy útil por ejemplo para gastos o ingresos fijos de cada mes, como el sueldo, la hipoteca, pagos de seguros, cuotas, etcétera).

Tiene otras funcionalidades muy interesantes como pueden ser la generación de gráficos o informes de cualquier operación. Hay multitud de opciones a la hora de generar estos gráficos y pueden exportarse a HTML. También está implementada la opción de hacer operaciones con una entidad bancaria de forma online (siempre que el sistema del banco soporte el estándar OFX o el HBCI). La importación de información financiera de otros programas (dependiendo del formato), el manejo de transacciones multi-divisas o la gestión de carteras de acciones/fondos de inversión son otras de las opciones disponibles.

El libro de cuentas puede ser guardado en un único archivo XML o en una base de datos, gracias a que GNUCash ha añadido esta posibilidad. También se pueden exportar a un fichero CSV el árbol de cuentas o las transacciones.

2.4.a) Base de datos

La versión 2.4.0. (publicada el 21 de diciembre de 2010) introdujo el motor gráfico WebKit y la posibilidad de operar usando bases de datos SQLite 3, MySQL o PostgreSQL además del usual formato de datos en XML.

Para ello, la comunidad desarrolló un conjunto de tablas que recogían la información necesaria para que el programa funcione y todos los datos que anteriormente se guardaban en XML, así como las relaciones entre los datos de las distintas tablas y los **GUID** que actúan como claves primaria o secundaria en cada caso. Esto dio como resultado el diagrama entidad-relación que se muestra en la Figura 1.

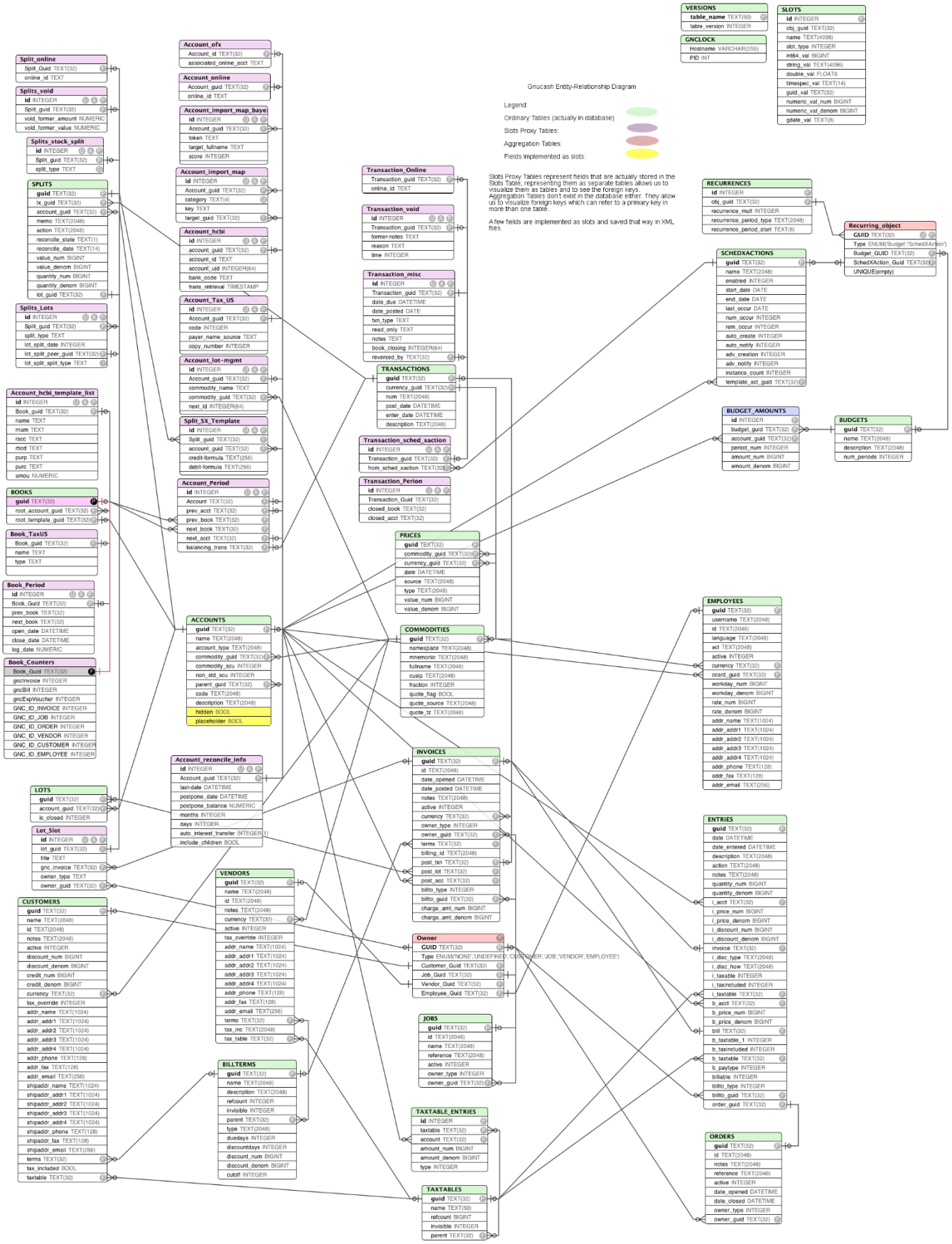

Figura 1. Diagrama Entidad-Relación de GNUCash

Sección 2.5 Otras tecnologías de desarrollo de aplicaciones web

2.5.a) PHP

Es un lenguaje de programación del lado del servidor orientado sobre todo al desarrollo web de contenido dinámico con acceso a información almacenada en una base de datos. Se puede integrar directamente en el HTML, evitando tener que llamar a un archivo externo que procese los datos. Un servidor web interpreta el código y genera la página web resultante. Normalmente se suele integrar con Apache como servidor web y MySQL como servidor de base de datos en entornos Linux en un paquete llamado LAMP, pero permite la conexión con diferentes tipos de servidores de bases de datos tanto SQL (PostgreSQL, Oracle, etc) como NoSQL (MongoDB, etc).

PHP tiene las ventajas de ser el más usado y por tanto el que tiene mayor comunidad y documentación detrás. También cuenta con varios CMS (sistemas de administración de contenido), es decir, herramientas que permiten editar, crear, clasificar o publicar cualquier tipo de información en una página web, actuando sobre una base de datos. Los más famosos son WordPress, Joomla y Drupal. Por otra parte cuenta con gran cantidad de librerías y frameworks que facilitan diferentes tareas. Los más importantes son frameworks que utilizan el modelo MVC (Modelo-Vista-Controlador) como Symfony, Laravel o Codeigniter.

Es un lenguaje más similar a C o a JavaScript que a lenguajes de marcas como HTML o XML. La principal diferencia con JavaScript es que éste se ejecuta en el navegador, mientras que PHP lo hace en el servidor. Por lo tanto puede acceder a cualquier recurso disponible en el servidor, como la base de datos. Además no requiere que el navegador del cliente sea compatible con PHP ya que la ejecución se hace en el servidor, que obviamente sí debe ser compatible, y el resultado se envía al navegador cliente, generalmente como una página en formato HTML o WML. Otra diferencia reseñable es que PHP es un lenguaje no tipado, es decir, sus variables no necesitan ser declaradas ni inicializadas.

2.5.b) Ruby on Rails

Es un framework para desarrollar aplicaciones web de código abierto, escrito en el lenguaje de programación Ruby. Sigue el patrón MVC (Modelo-Vista-Controlador) y está diseñado para hacer más fácil la programación de aplicaciones web. Permite que escribiendo menos código se realice más que en otros lenguajes o frameworks, siempre que lo hagas según "el modo Rails", es decir, que existe una forma que es mejor que las demás para hacer las cosas, y te permite ser más productivo en menos tiempo. Está basado en el principio DRY (Don't repeat yourself), que significa que repetir código es una mala práctica y no se debe hacer. También se basa en otro principio "Convención sobre programación", que significa que en lugar de requerir que especifiques toda la configuración, Rails hace suposiciones sobre lo que quieres hacer y cómo vas a hacerlo.

2.5.c) Bottle

Bottle es un microframework muy simple desarrollado en Python que proporciona un mínimo de herramientas al desarrollador (enrutamiento, plantillas y una pequeña abstracción sobre WSGI). Está diseñado para ser rápido sencillo y ligero, ideal para desarrolladores que buscan flexibilidad, crear una API web, o programar algo realmente simple, con facilidad y rapidez.

Soporta URL con parámetros, plantillas, base de datos clave-valor y complementos para muchos WSGI de terceros y plugins para bases de datos popuares. Puede ejecutarse como servidor web independiente.

2.5.d) Flask

Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un número mínimo de líneas de código. Está basado en la especificación WSGI, el motor de templates Jinja2 inspirado en Django y tiene una licencia BSD.

2.5.e) Pyramid

Pyramid es un framework web de código abierto escrito en Python y basado en WSGI. Es minimalista, inspirado en Zope, Pylons y Django. Forma parte del Pylons project. Está integrado tanto con bases de datos

SQL a través de SQLAlchemy, como con bases de datos de objetos Zope y otros NoSQL como CouchDB. Permite definir rutas usando expresiones regulares que se asignan a objetos. Permite también recorridos jerárquicos de objetos, siendo cada parte de la URL un objeto que contiene otros objetos, quedando parecida a la ruta de una carpeta en un sistema de archivos.

Sección 2.6 Otras aplicaciones de contabilidad

2.6.a) Quicken

Quicken es una herramienta de gestión de finanzas personales desarrollada por la empresa Quicken Ink (recientemente vendida por Intuit Inc a HIG Capital). Se ejecuta desde sistemas Windows y Macintosh. Las versiones iniciales, se lanzó por primera vez en 1983, se ejecutaban en DOS. Existen diferentes versiones de Quicken con más o menos características u orientadas a otras gestiones financieras como alquileres de viviendas.

Es líder en el mercado de América del Norte pero las funciones básicas se utilizan ampliamente en otros países, habiendo sido adaptadas las versiones para una gran variedad de mercados.

Quicken Online era una versión gratuita, alojada en los servidores de Intuit. La empresa almacenaba los datos del usuario y proporciona parches y actualizaciones automáticas del software de forma periódica. Posteriormente se lanzó como una suscripción de pago mensual. Durante el año de gratuidad consiguió más de 1,5 millones de clientes. Su desarrollo se detuvo en 2010 tras la compra de Mint.com

2.6.b) Mint.com

Mint.com es quizás la alternativa más similar a lo desarrollado en este proyecto. Gratuito, basado en web, es un gestor de finanzas personales desarrollado por Aaron Patzer. Su área de servicio es Estados Unidos y Canadá, donde tiene acuerdos para la agregación de cuentas. Inicialmente con Yodlee pero actualmente utiliza Intuit para conectar las cuentas bancarias. Permite por tanto rastrear los movimientos, transacciones, saldos y préstamos a través de una única interfaz de usuario así como crear presupuestos y establecer metas.

Según últimos datos, Mint.com afirmó tener más de 20 millones de usuarios en 2016, el doble que tres años antes.

Su punto débil más importante es la seguridad. Para su funcionamiento es necesario proporcionar los nombres de usuario y contraseñas de acceso a las oficinas electrónicas de los bancos del usuario. Estos datos son almacenados en las bases de datos de Mint.com en un formato desencriptable. Lo cual plantea preocupación ante posibles ataques que podrían poner esa información sensible al acceso de terceras personas malintencionadas. Algunos bancos proporcionan códigos de acceso alternativos con acceso de sólo lectura a la información financiera para el uso de esta aplicación. Reduce el riesgo de que un ser malintencionado, al obtenerlos, pudiera operar en las cuentas del usuario de Mint.com, pero sigue planteando un problema de privacidad en tanto que los datos económicos quedarían al descubierto.

2.6.c) MyValue

MyValue es una aplicación que se asemeja mucho a GNUCash. Fundada en España en 2009, financiada por el CDTI del Ministerio de Ciencia e Innovación. Se conecta a los bancos del usuario para obtener la información actualizada de los movimientos en sus cuentas (esto lo permite GNUCash en algunos bancos, aunque normalmente se hará este paso de información a mano). MyValue permite además tener una cuenta de efectivo en la que ir anotando los gastos realizados con dinero en metálico y que por tanto no quedan reflejadas en las cuentas bancarias. Esta es la característica que nos motiva a introducirla en el estado del arte y que la diferencia de otra conocida aplicación, fintonic, que únicamente recopila los datos de los bancos, sin permitir al usuario introducir movimientos de forma manual.

En la aplicación, que es multiplataforma (versión web para escritorio y aplicaciones móviles para Android y iPhone), se pueden categorizar movimientos según las preferencias del usuario, controlar recibos y facturas, hacer presupuestos que se ajusten a cada necesidad, fijar objetivos de ahorro y monitorizar inversiones.

Capítulo 3 Aplicación web

Sección 3.1 Objetivos

El objetivo de la aplicación web de GNUCash, es el de ofrecer un servicio web accesible a través de cualquier plataforma, mediante el que se puedan llevar de una forma sencilla las finanzas personales. Pudiendo consultar y modificar nuestras cuentas tan sólo necesitando un dispositivo con acceso a Internet. Utilizando la nube para el almacenamiento de estos datos.

Se aprovechará que se ha introducido en GNUCash la posibilidad de almacenar la información en una base de datos externa, en lugar del XML local. En este caso se utiliza MySQL como servidor de bases de datos. De esta forma, se almacenarán los datos en la nube y serán accesibles desde internet.

Se pretende permitir además que pueda haber varios usuarios conectados a la misma base de datos, o el mismo usuario en distintos dispositivos. Las vistas y los cambios que se realicen en cualquiera de estas sesiones, se reflejarán en la base de datos y se actualizarán de forma inmediata. Esto favorece que la aplicación web pueda usarse de forma simultánea en distintos dispositivos y que los datos contenidos siempre sean consistentes.

Se mejorará así la alternativa de sincronizar el XML ya que si en ese caso hubiera varios usuarios modificando de forma simultánea, se crearía un conflicto con los ficheros XML y sólo perdurarían los datos del último fichero guardado, perdiéndose las modificaciones de los demás.

El desarrollo de esta aplicación responde a la escasez de opciones en el mercado, libres y gratuitas, que permitan al usuario llevar sus cuentas tanto desde un ordenador como desde un Smartphone o una Tablet sin necesidad de descargarse una aplicación en cada dispositivo.

Sección 3.2 Tecnologías utilizadas

3.2.a) Servidor Web

En este proyecto se utilizan las tecnologías que se describen a continuación. Se alojan en un servidor web con Sistema Operativo Ubuntu server. Todas las tecnologías están disponibles en el Sistema de Gestión de Paquetes (APT) por lo que, aunque también se encuentran en paquetes "amigables para el usuario" como LAMPP, `apt-get` hace que las aplicaciones se integren mejor en el sistema y se comuniquen e integren entre sí como necesitan para funcionar.

En los próximos apartados se indicarán algunos comandos necesarios para la instalación y uso en un servidor de estas características. Las instrucciones para máquinas con otros Sistemas Operativos podrían ser diferentes.

3.2.b) Django

A la hora de elegir las tecnologías que se iban a utilizar para desarrollar esta aplicación web, se buscó un lenguaje de programación o framework que no se estudie durante el grado y con el que se amplíen los conocimientos adquiridos hasta la fecha. Aprender desde cero como utilizarlo y adaptarlo a las necesidades del proyecto es un buen mecanismo de aprendizaje. La idea era apartar un poco el uso de los lenguajes vistos usualmente a lo largo de la carrera (PHP, Java), y explotar la capacidad de desarrollar una aplicación web con una tecnología totalmente nueva para alguien que está a punto de graduarse. Al tratarse de una aplicación web CRUD, es decir, basada sobre todo en la modificación constante de una base de datos, se buscan framework apropiados para esta tarea. En un principio, se barajaron dos opciones. Ruby on Rails (escrito en Ruby), y Django (escrito en Python).

Finalmente se decidió Django sobre todo porque está escrito en Python, un lenguaje que llama la atención y que viene bien aprender, ya que puede ser muy útil debido a sus características y facilidades. Como por ejemplo a la hora de convertir tipos o modificar variables, ya que cuenta con funciones propias que podían sernos muy útiles durante el desarrollo. Además es un lenguaje sencillo de aprender. Por otra parte, a la hora de hacer consultas a la base de datos, Django cuenta con un ORM (Object-Relational Mapping), que es un modelo de programación que consiste

en la transformación de tablas de la base de datos en una serie de entidades que simplifican las tareas de acceso a los datos para el programador. En el caso de Django, se crean unos modelos (tipos de datos) que representan las tablas de la base de datos, y que permiten hacer consultas SQL sin utilizar SQL y con simples funciones preparadas para ello.

Para este proyecto se utiliza las versiones Django 1.8.5 y Python 2.7.9. La instalación de estas tecnologías se realiza mediante la siguiente secuencia de comandos:

```
# apt-get install python
# apt-get install python-pip python-dev build-essential
# pip install --upgrade pip
# pip install django==1.8.5
```

3.2.c) MySQL y phpMyAdmin

Para elegir el sistema gestor de base de datos hay que tener en cuenta que GNUCash da la posibilidad de utilizar bases de datos SQLite3, MySQL o PostgreSQL. Desde un principio, la idea era utilizar MySQL, ya que es muy conocido y ampliamente utilizado. Se consideró por tanto que era la mejor solución para nuestra aplicación web, ya que es sencillo de utilizar y hay mucha documentación y una gran comunidad detrás. Además la integración con Django permitiría trabajar cómodamente.

A esta elección le acompaña la posibilidad de utilizar phpMyAdmin, un gestor web de bases de datos que, aunque principalmente se utilizará la propia aplicación, phpMyAdmin ha podido servir para comprobar el comportamiento del código y cómo la modificación de modelos Django afectaba y se reflejaba en la base de datos MySQL correctamente.

El uso de MySQL requiere instalar una compatibilidad con Python. Esta compatibilidad y todo el paquete se consigue con los siguientes comandos:

```
# apt-get install python-mysqldb
# apt-get install mysql
# apt-get install mysql-client
# apt-get install mysql-server
# apt-get install phpmyadmin
```

Una vez instalado, será necesario arrancar el servicio MySQL. A continuación se indican los comandos necesarios para arrancar, detener o reiniciar el servicio.

```
$ service mysql start
$ service mysql stop
$ service mysql restart
```

Sección 3.3 Metodología de trabajo

En esta sección se explica cómo se ha ido desarrollando la aplicación y los pasos previos de pruebas y reconocimiento del entorno, ya que aunque Python es un lenguaje sencillo, nunca habíamos trabajado con él, ni con el framework Django.

3.3.a) Máquinas virtuales

Para comenzar a trabajar se crearon entornos de prueba en máquinas virtuales con el mismo sistema operativo que tendría el servidor que alojaría finalmente la aplicación. En estas máquinas se lleva a cabo la instalación de las tecnologías elegidas. En un primer momento se utilizó el paquete LAMPP, pero visto que generaba algunos problemas se decidió instalar todo lo necesario a través de los repositorios del APT. Una vez montado el entorno en estas máquinas virtuales empezó el proceso de familiarización con él y con el lenguaje, creando aplicaciones de prueba, a menor escala, que tuvieran una funcionalidad similar a la que se buscaba para el proyecto. Finalmente se comenzaron a hacer las primeras pruebas con aplicaciones que integraban la base de datos de GNUCash y la modificaban, comprobando el comportamiento de la aplicación de escritorio tras estos cambios externos.

3.3.b) Conexión al servidor

El grueso del proyecto ha sido desarrollado directamente en el servidor. Las conexiones al mismo se hacen a través de SSH con autentificación por claves públicas. La conexión es necesaria para arrancar el servidor ya que durante la implementación no estaba siempre en marcha. La aplicación no ha estado abierta al público y su ejecución se ha hecho siempre en local, pero redirigiendo los puertos correspondientes a través de un túnel seguro SSH para cubrir la carencia de otro tipo de cifrado.

De esta forma, se puede trabajar viendo los resultados de forma remota, pero a su vez, no están públicos en internet ni los datos viajan de manera insegura. Estableciendo la conexión con el siguiente comando, en una máquina con clave privada autorizada obviamente, redirigimos la ejecución del servicio Django de la aplicación al navegador local:

```
ssh -L 8080:localhost:8000 tfg15@isabelita.fdi.ucm.es
```

El atributo –L redirige el puerto indicado (8000) de la ejecución en la máquina remota al puerto indicado (en este caso el 8080) del dominio localhost. De esta forma se puede acceder al resultado de arrancar el servidor de la aplicación web. La aplicación se arranca mediante el siguiente comando, que debe invocarse desde la carpeta del proyecto, en la que se encuentra el fichero manage.py:

```
python manage.py runserver
```

Por defecto, se lanza el servidor en la dirección http://localhost:8000, a la que se puede acceder remotamente gracias a la redirección explicada en el comando anterior.

También está disponible el acceso a la base de datos a través de phpMyAdmin vía web (http://localhost/phpmyadmin). La modificación de ficheros podía hacerse de forma remota gracias a que todo el proyecto está en una carpeta sincronizada y compartida por Dropbox.

3.3.c) Control de versiones

Como se indica en el último párrafo del apartado anterior, el proyecto está en una carpeta sincronizada con Dropbox, por lo que cualquier cambio en uno de sus ficheros queda reflejado en el control de versiones que incorpora Dropbox. Si bien este control no es real de versiones de la aplicación, sino de versiones de cada fichero concreto, que pueden ser menores y no tener completo un funcionamiento correcto en pasos intermedios de la implementación.

El verdadero control de versiones *major* de la aplicación, con cambios sustanciales entre una y otra y funcionalidades definidas añadidas a cada una se ha realizado a través de github. El proyecto completo puede encontrarse en https://github.com/ugedo/gnucashdjango

3.3.d) Reuniones periódicas

Como norma general, desde el inicio del proyecto, el equipo de trabajo se ha reunido quincenalmente con el director del proyecto para poner en común los avances, dudas y problemas, así como para definir los siguientes pasos a seguir. En algunos periodos de máxima actividad, las reuniones se han efectuado de forma semanal. Especialmente en los últimos meses de implementación.

Respecto a los componentes del equipo, ambos han estado constantemente en comunicación para poner en común los distintos apartados desarrollados, los conocimientos adquiridos durante la etapa de aprendizaje y para repartir las tareas a realizar.

Capítulo 4 Implementación

Tras haber detallado las tecnologías que se han usado, la arquitectura básica de esta aplicación y la metodología de trabajo, en este capítulo se explicará cómo está estructurado el proyecto y se hablará de los detalles a la hora de implementarlo.

Sección 4.1 Estructura del proyecto

Lo primero que hay que tener en cuenta al hablar de qué estructura sigue este proyecto, es que se trata de una aplicación en Django, por lo tanto sigue un patrón definido respecto a la arquitectura. Es el patrón MVT (Modelo-Vista-Template). Este patrón lo que hace es separar la aplicación en capas.

Modelos: existe una capa de datos, que corresponde a los modelos. Estos constituyen la representación de los datos de la aplicación. Contiene los campos básicos y el comportamiento de los datos que será almacenado. Por lo general, y es así como sucede en esta aplicación, cada modelo se convierte en una tabla de la base de datos. Django proporciona el acceso a la base de datos automáticamente a través de los modelos. Los campos tendrán un tipo y podrán aplicarse opciones, de entre los definidos en la documentación de Django. Cada modelo es una subclase de **django.db.models.Model** y suelen declararse en el fichero **models.py**

Vistas: la capa de proceso en Django se corresponde con las vistas, en las que se declaran las funciones que procesan los datos y en las que se realizan las modificaciones y cálculos necesarios para cambiar los modelos. Hacen una consulta Web (*request*) y se devuelve una respuesta Web (*response*) a los *templates* y éstos lo muestran como código HTML.

Plantillas: corresponden con la capa de presentación. Supone una separación entre la lógica de programación y la lógica de presentación. Lo que hacen las plantillas de Django es traducir código Python de modo que se puedan transferir datos desde las vistas en forma de código HTML, que es accesible desde la web, mediante los *template tags*.

Esta aplicación web se compone de los siguientes ficheros y directorios, representados como un árbol de directorios, que se detalla a continuación y que da una visión global completa del proyecto y de la separación entre las distintas capas de la arquitectura de las que se ha hablado anteriormente.

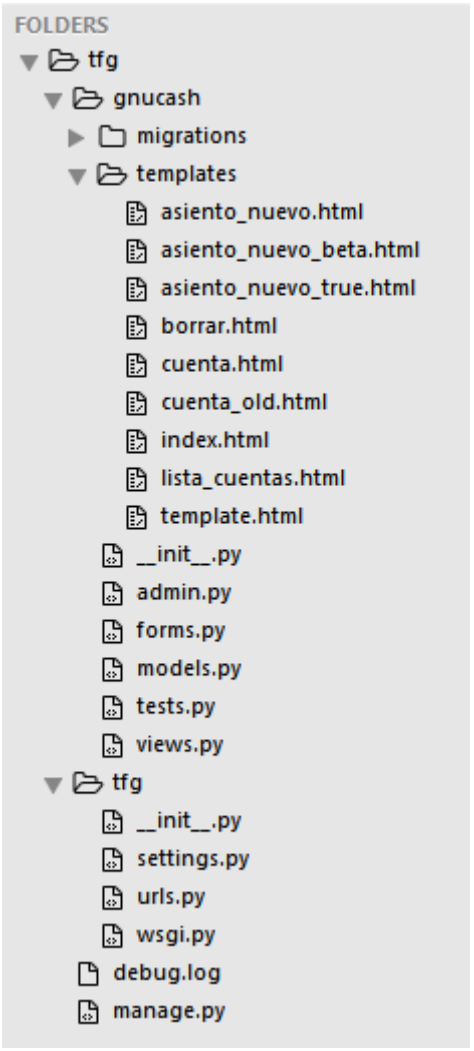

Figura 2. Árbol de directorios

En Django, cada proyecto contiene una configuración, pero puede contener varias aplicaciones independientes cada una con sus modelos, vistas y platillas. Dentro del proyecto "tfg", se encuentra otra carpeta "tfg", que contiene la configuración del proyecto, y una carpeta "gnucash", que es la aplicación, dentro de la cual se encuentran los modelos, vistas y plantillas.

4.1.a) Configuración del proyecto

settings.py: El primer paso al crear el proyecto en Django fue configurarlo para poder usarlo con una base de datos MySQL. Para ello es preciso modificar el archivo settings.py, especificando el tipo de servidor de base de datos, el nombre de la base de datos ("gnucash"), usuario, contraseña, host y puerto. También es necesario configurar otro tipo de cosas, como registrar el nombre de la aplicación que se va a ejecutar o dónde se encuentra la carpeta con las plantillas de la aplicación.

Para la base de datos, se configura el diccionario `DATABASES` con los datos requeridos. En este caso cabe destacar que el campo `ENGINE` debe definirse como `django.db.backends.mysql` para MySQL.

debug.log: También se creó un sistema de logger en el proyecto (se configura en settings.py), que permitiese depurar y encontrar errores durante el desarrollo. Según el nivel especificado, se escriben los errores en caso de existir. Se pueden comprobar el valor de variables u objetos desde cualquier parte del código simplemente invocando al log, que lo escribirá todo en este archivo que se encuentra en la raíz del proyecto.

4.1.b) Estructura de la aplicación

A continuación se explica qué son y el uso que se le da en este proyecto a los ficheros importantes que encontramos dentro de la aplicación.

models.py: Es el fichero en el que se encuentran los modelos. Cada modelo es una clase en Python que representa cada tabla de la base de datos. Los modelos contienen los campos de cada tabla con su formato específico.

En el fichero **models.py** de esta aplicación existen veinticuatro clases, que son las veinticuatro tablas correspondientes en la base de datos de GNUCash. Como se puede observar en la Figura 3, que se corresponde con el modelo de la tabla **Accounts**, el modelo es una clase de Python, en la que se define cada campo de la tabla con un tipo que hereda de *models* de forma que Django lo pueda interpretar.

```python
class Accounts(models.Model):
    guid = models.CharField(primary_key=True, max_length=32)
    name = models.CharField(max_length=2048)
    account_type = models.CharField(max_length=2048)
    commodity_guid = models.CharField(max_length=32, blank=True, null=True)
    commodity_scu = models.IntegerField()
    non_std_scu = models.IntegerField()
    parent_guid = models.CharField(max_length=32, blank=True, null=True)
    code = models.CharField(max_length=2048, blank=True, null=True)
    description = models.CharField(max_length=2048, blank=True, null=True)
    hidden = models.IntegerField(blank=True, null=True)
    placeholder = models.IntegerField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'accounts'
```

Figura 3. Modelo de la tabla Accounts

Así pues, por ejemplo el nombre de la cuenta (*name*) es un campo de texto de longitud máxima 2048 caracteres, lo que se corresponde con la secuencia:

```
name = models.CharField(max_length=2048)
```

De la misma forma, se definen el resto de campos de acuerdo al estándar de los modelos de Django. En este mismo ejemplo puede verse que los enteros se definen como `IntegerField`, o que la clave primaria del modelo se indica con la propiedad `primary_key=true`.

Los modelos pueden contener la clase interna **`Meta`**, que sirve para dar metadatos al modelo, es decir, especificar todo lo que no es un campo, ya sean opciones de ordenamiento o el nombre de la tabla de la base de datos.

En el ejemplo, se especifica el nombre de la tabla de base de datos con la opción `db_table='accounts'`, y de acuerdo a la documentación de Django se indica la opción `managed=False` para especificar a Django que no administre la creación, modificación o eliminación de la tabla. Se dejará esta tarea en manos de la aplicación oficial de GNUCash.

views.py: Aquí se encuentran todas las vistas del proyecto y funciones auxiliares utilizadas para hacer cálculos. Hay que recordar que las vistas son el núcleo programático de la aplicación. Las vistas están asociadas a una URLs. Esto se hace en un archivo que se encuentra dentro de la configuración del proyecto llamado **urls.py**. En este archivo se asocian los nombres de las vistas a una URLs concreta dentro del proyecto. En las

vistas se tratan los datos que posteriormente renderizamos en las plantillas como HTML que se mostrará en las URLs especificadas.

En la Figura 4 se muestra como se asocia en el archivo **urls.py** las direcciones URL relativas a la dirección de ejecución del entorno, con la vista que se desea ejecutar al visitar la URL correspondiente. Las direcciones están escritas como expresión regular.

```
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'gnucash.views.listaCuentas'),
    url(r'^asiento_nuevo.html$','gnucash.views.post_new'),
    url(r'^asiento_nuevo_true.html$','gnucash.views.guardar_asiento'),
    url(r'^lista_cuentas.html$','gnucash.views.listaCuentas'),
    url(r'^cuenta.html$','gnucash.views.verCuentayAsiento'),
    url(r'^asiento_cuenta.html$','gnucash.views.verCuentayAsiento'),
    url(r'^borrar$','gnucash.views.borrarAsiento'),
]
```

Figura 4. Direcciones URL y vistas asociadas

Tomando como ejemplo la vista `listaCuentas`, en la Figura 5 se muestra cómo funciona y cómo se implementa una vista en Django.

```
def listaCuentas(request):
    book = Books.objects.all()[:1]
    root = Accounts.objects.get(guid=book[0].root_account_guid)
    account_list = listaCuentas_rec('', root)
    account_list.sort()
    i = 1
    account_toHTML = list()
    amounts = list()
    while i < len(account_list):
        account_list[i] = account_list[i][0].split(':')[1:], account_list[i][1]
        account_toHTML.append(('     ' *
            (len(account_list[i][0])-1) +
            '<i class="fa fa-university" aria-hidden="true"></i>' + ' ' +
            '<a href="/cuenta.html?guid=' + account_list[i][1] + '">' +
            account_list[i][0][-1] + '</a><br/>',totalizarCuenta(account_list[i][1]),
            Accounts.objects.get(guid=account_list[i][1]).description))
        i += 1
    return render_to_response('lista_cuentas.html', {'account_list': account_toHTML})
```

Figura 5. Código de la vista "listaCuentas"

Como se puede apreciar, una misma vista puede estar referenciada por distintas URLs, y éstas no tienen por qué coincidir con el nombre de la plantilla en la que retorna el resultado de ejecución de la vista.

Una vista es una función Python, que recibe el entorno por `request`, y devuelve un conjunto de datos a una plantilla (**lista_cuentas.html** en este caso) por el método `render_to_response`. Los datos se convierten en

un objeto **httpResponse** con el contenido de la página solicitada. En la vista `listaCuentas`, se obtiene la lista de cuentas mediante consultas a la base de datos y se van introduciendo en una lista en Python que se envían como argumento a la plantilla para que los muestre en la página.

templates: Esta carpeta contiene todas las plantillas HTML asociadas a URLs concretas. La plantilla **index.html** podría ser la pantalla inicial de la aplicación, muestra una lista de las operaciones implementadas desde la que se llamará a otras templates como **asiento_nuevo.html**, que es la pantalla en la que se muestra un formulario para crear un nuevo asiento e insertarlo en base de datos. También **lista_cuentas.html**, que muestra un listado con todas las cuentas del libro de cuentas, con su balance total. En **cuenta.html** se podrá ver al detalle cada cuenta con sus movimientos, cuentas destino, descripción, cantidades y fecha. Los *templates* en Django pueden extender a otros *templates*, característica que se utiliza por ejemplo con **template.html**, que es utilizado como cabecera para otras plantillas.

Siguiendo con el ejemplo explicado en el apartado de **views.py**, se muestra la parte importante de la plantilla **lista_cuentas.html**.

```
1  {% extends "template.html" %}
2  {% block titulo %} Lista de cuentas {% endblock titulo %}
3  {% load humanize %}
4  {% block contenido %}
5  <table>
6      <tr class="cabecera">
7          <td><h4>Nombre de cuenta</h4></td>
8          <td><h4>Descripción</h4></td>
9          <td align="right"><h4>Total</h4></td>
10     </tr>
11
12 {% for account in account_list %}
13     <tr>
14         <td>{{account.0|safe|escape}}</td>
15         <td>{{account.2}}</td>
16         <td align="right">{{account.1|floatformat:2|intcomma}} €</td>
17     </tr>
18 {% endfor %}
19 </table>
20 {% endblock contenido %}
```

Figura 6. Código de la plantilla "lista_cuentas.html"

Se trata de un archivo HTML, en el que se introducen las sentencias Django entre llaves, con la sintaxis "`{% sentencia %}`" y las variables que llegan desde la vista entre dos llaves, con la sintaxis "`{{ variable }}`". En el ejemplo mostrado, y en general en todas las plantillas de la aplicación, se extiende de la plantilla **template.html** personalizando el título (Lista de cuentas), y el contenido propio en HTML, mostrando la lista de cuentas con un bucle `for` de `account_list`, que es la lista recibida desde la vista. De esta lista se accede a los datos concretos utilizando "`.`" (punto) para determinar la posición correspondiente. Se pueden utilizar filtros de plantillas para alterar el valor de las variables como por ejemplo al mostrar el nombre de las cuentas:

```
{{ account.0|safe|escape }}
```

Para escapar de forma correcta el código HTML que se ha introducido en la vista. O para mostrar dos decimales en el balance, que es el formato utilizado por GNUCash y el más habitual para importes monetarios:

```
{{account.1|floatformat:2|intcomma}}
```

En las plantillas es también donde se introducen los estilos de la visualización de la página HTML con código CSS, buscando que se parezcan lo máximo posible al programa de escritorio.

```
<style type="text/css">
    body {
        background: #fbeee6;
    }
    .cabecera {
        background: #fae5d3 !important;
    }
    .ultimaCol {
        width: 50%;
    }
    table {
        border-collapse: collapse;
        border: 1px;
        width: 100%;
    }
    tr:nth-child(even) {
        background: #fff
    }
    tr:nth-child(odd) {
        background: #fcf3cf
    }
    a:link {
        text-decoration: none;
        color: black;
    }
</style>
```

Figura 7. Hoja de estilos de "lista_cuentas.html"

forms.py: Contiene la definición de los formularios utilizados para solicitar información al usuario. Django ofrece una manera sencilla y útil para la creación de formularios dentro de la aplicación. Consiste en crear un archivo **forms.py** en el que se declaran los formularios importando los modelos de los que dependen dichos formularios. En **views.py** se crea la función que se encarga de realizar (en este caso) la inserción en base de datos. Desde la vista se le pasará el formulario al *template* para que lo muestre como HTML. En el de esta aplicación **forms.py** se crea el formulario que permite guardar un asiento en base de datos. Además de dos funciones que permiten que la elección de cuenta sea un desplegable con opciones en lugar de un campo de texto.

```
class FormularioAsiento_t(ModelForm):
    post_date = DateField(widget= DateInput(format=('%d-%m-%Y')), label=u'fecha')
    class Meta:
        model = Transactions
        fields = ('post_date', 'num', 'description',)
        labels = {
            'num': ('Num'),
            'post_date': ('Fecha'),
            'description': ('Descripci&oacute;n'),
        }

class FormularioAsiento_s(ModelForm):
    options = listaCuentasC()
    account_guid = ChoiceField(widget=Select, label=u'Cuenta Origen', choices = options)
    reconcile_state = ChoiceField(widget=Select, label=u'Reconcile?', choices = reconcile)
    class Meta:
        model = Splits
        fields = ('account_guid', 'reconcile_state', 'value_num', )
        labels = {
            'account_guid': ('Cuenta Origen'),
            'reconcile_state': ('&iquest;Reconcile?'),
            'value_num': ('Valor'),
        }
```

Figura 8. Algunos formularios del fichero "forms.py"

Como puede observarse, los formularios también son clases de Python. En ellas está contenida una clase Meta que es donde se indica el modelo en el que se basa el formulario, se seleccionan los campos que se esperan obtener mediante el formulario y se modifica si se desea algunos valores como las etiquetas que se mostrarán al usuario en cada campo.

Sección 4.2 Base de datos

Este proyecto trata de complementar una aplicación existente con una base de datos ya existente, por este motivo, es necesario obtener esa base de datos directamente de la aplicación GNUCash. Para ello, se crea un libro de contabilidad nuevo y vacío desde la aplicación de escritorio y se guarda en el servidor MySQL.

Una vez obtenida la base de datos MySQL, vacía pero con la estructura que necesita el programa, debe ser integrada en la aplicación, esto se hace mediante la creación de los modelos correspondientes a cada una de las tablas.

Django ofrece una facilidad para crear de forma automática los modelos a partir de una base de datos existente. Lo primero que hay que hacer es, por supuesto, conectar la aplicación que estamos desarrollando con la base de datos, como hemos indicado anteriormente, en **settings.py**.

Una vez creado el entorno, se lanza la ejecución del siguiente comando desde la carpeta en la que se encuentra el entorno

```
python manage.py inspectdb > myapp/models.py
```

Tras su ejecución, se crean en models.py las veinticuatro clases correspondientes a las tablas de la base de datos.

Sobre estos modelos generados automáticamente, es necesario hacer una serie de modificaciones en ellos para que funcionen de manera correcta.

- Django agrega automáticamente un campo *id* de clave primaria si el modelo no lo tiene. Habrá que quitar esas líneas y establecer como clave primaria el campo de GUID correspondiente.
- El comando inspectdb no puede detectar si un campo es autoincrementado, así que habrá que cambiar a AutoField los que corresponda.
- Hay que comprobar que los tipos de cada columna (VARCHAR, DATE...) se han convertido correctamente a tipos de cada campo del modelo (CharField, DataField...)

El siguiente paso será crear las tablas del núcleo de Django, esto se consigue con el comando

```
python manage.py migrate --fake
```

Así se sincroniza el estado de la base de datos con el conjunto de modelos. Al hacerlo, Django modifica el nombre de las tablas en MySQL añadiendo el prefijo con el nombre de la aplicación, de forma que la tabla **Split**, por ejemplo, pasa a llamarse **gnucash_Splits**. Eso no es lo que se quiere para esta aplicación ya que al cambiar el nombre de las tablas, éstas no serían reconocidas por la aplicación de escritorio GNUCash y la intención es que sí lo sea. Es por ello por lo que se añade la opción `--fake`.

Por otra parte, a pesar de tener veinticuatro tablas en la base de datos, este proyecto está enfocado en las funcionalidades más básicas de GNUCash, buscando ofrecer la posibilidad de agregar un asiento nuevo a la cuenta correspondiente en la base de datos. En general, hay muchas tablas que son para funciones específicas que incluso desde la aplicación de escritorio, un usuario particular no llegaría a utilizar. Las funcionalidades implementadas requieren únicamente el uso de tres tablas, que son las que se muestran a continuación.

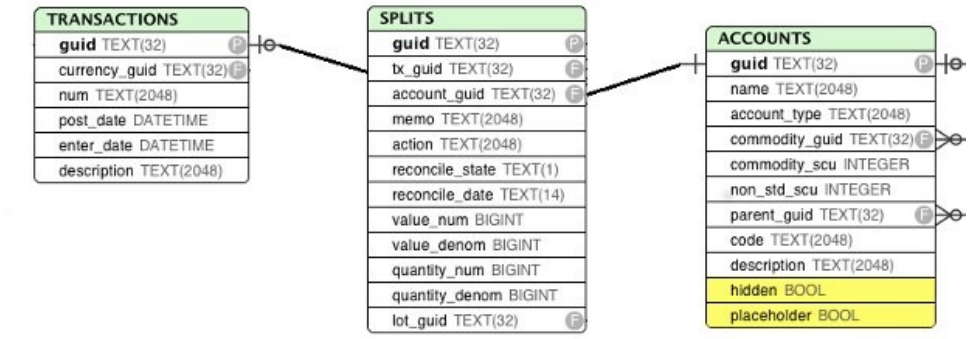

Figura 9. Tablas utilizadas de la base de datos

- **Accounts**: es la tabla que contiene las cuentas de nuestro libro de cuentas, en la que se almacena su nombre, su GUID, su tipo, la cuenta padre o su descripción.
- **Transactions**: contiene todas las transacciones, que son los movimientos que se producen cuando se mueve dinero entre dos cuentas. Se almacena entre otros, su GUID, su descripción y la fecha en la que se produjo. Al crearse una transacción se crean dos Splits asociados a cada cuenta en la que se produce el movimiento.
- **Splits**: en esta tabla se almacenan todos los Splits, asociados a una transacción y a una cuenta. Se almacena su GUID, el GUID de la cuenta, el GUID de la transacción, y su valor, entre otros datos.

Sección 4.3 Funcionamiento de las vistas

Las vistas, recogidas en el fichero **views.py**, es el núcleo programático de la aplicación. En ellas se encuentran todas las funciones que hacen posible el funcionamiento de la aplicación. Desde aquí se indica cómo han de calcularse los valores, recuperar datos de la base de datos y salvar los nuevos. Las funciones implementadas en **views.py** son lanzadas cuando el usuario navega por la URL que la referencia. Esto activa el procedimiento programado en la vista y manda los resultados a la plantilla HTML que es la encargada de mostrar finalmente la página web al usuario.

4.3.a) Lista de cuentas

Es la vista principal del proyecto accesible a través del *índex* de la aplicación. En ella se producen los cálculos y consultas a la base de datos necesarios para que se muestren posteriormente como HTML en la plantilla **lista_cuentas.html**.

Se va a detallar paso a paso el proceso seguido para implementar la vista y crear una lista de cuentas de GNUCash.

El primer paso es mediante una consulta a la tabla **books** de la base de datos, obtener el libro de cuentas necesario, (de momento contamos con tener un único libro) y mediante su GUID, consultar en la tabla **accounts** para conseguir la cuenta raíz del libro de cuentas.

A partir de la cuenta raíz se buscan todos las cuentas hijas y se guardan en una tupla el nombre completo de la cuenta, con su ruta de padres incluida, y su GUID. Para obtener ese nombre completo, se utiliza una función recursiva que va obteniendo para cada cuenta, las que son sus hijas y concatenando los nombres separados por dos puntos (:). De esta forma, la cuenta “Metálico” hija de “Activo Circulante” hija de “Activo” quedará representada como “Activo:Activo Circulante:Metálico”.

Por cada cuenta calculamos el total de sus movimientos, mediante la función `totalizarCuenta` que a partir del GUID, obtiene todos los Splits de esa cuenta. El valor de cada asiento se va sumando (o restando, según el tipo de cuenta) a la variable `amount`. Tras esta operación es necesario recuperar los hijos de la cuenta y llamar de forma recursiva a la misma función con el GUID de cada uno de sus hijos, para obtener el total de toda la categoría.

Se hace un tratamiento a la lista para pasársela a la plantilla de forma que tabule las cuentas dependiendo del número de padres que tengan. También se añaden a esta lista los enlaces en código HTML con el GUID de las cuentas para pasárselo por GET a la vista `verCuenta`, para saber de qué cuenta tiene que mostrar los detalles. Por último se añade a la lista el total de la cuenta calculado anteriormente.

Esta lista es enviada mediante el método `render_to_response` a la plantilla **lista_cuentas.html**, que al estar mapeada con esa vista en el archivo **urls.py**, nos mostrará en dicha URL el código HTML de la plantilla. En la plantilla simplemente recorremos la lista de cuentas mostrando en una tabla las cuentas, en las que se puede clicar para enlazar a ver su detalle, la descripción de la cuenta, y el total de cada cuenta. Se muestra tabulado como un árbol de directorios, siendo esta muy similar a la forma que utiliza GNUCash para mostrar las cuentas.

4.3.b) Guardar asiento

Inicialmente se creó una función que mostraba un formulario el cual pedía los datos necesarios para un asiento nuevo, así como las cuentas origen y destino del mismo. El formulario consta en realidad de tres formularios definidos en el fichero **forms.py**. Uno para los datos de Transaction y otros dos para los Splits de cada cuenta. Los datos de los Splits son compartidos salvo el GUID de cada cuenta, por lo que ese último formulario sólo muestra un `select` con las cuentas. Para mostrar los formularios, en la plantilla se utiliza `form.as_p`, que lo muestra como párrafo (etiqueta `p` de HTML) o podría utilizarse `form.as_table`. En la función `post_new` están todas las operaciones que se realizan para tratar los datos recibidos del formulario y guardarlos en la base de datos.

4.3.c) Ver cuenta

Esta vista es la que realiza las consultas y modificaciones para mostrar los detalles de cada cuenta que se encuentra alojada en el libro de cuentas. Es decir, los movimientos que se han producido a lo largo del tiempo en esa cuenta, su descripción y su valor.

Se encuentra mapeada para mostrarse en la URL /cuenta.html, que se corresponde con la plantilla **cuenta.html** que es a la que le pasamos los datos desde la vista. Se accede a través de la lista de cuentas, y la vista siempre recibe por GET el GUID de la cuenta que se ha clicado desde la

lista, para así saber en esta vista de que cuenta tiene que mostrar los detalles.

Lo primero es llamar a la tabla Splits para obtener todos los movimientos que se han producido en esa cuenta. Recorriendo los Splits y realizando varios cálculos de cara a mostrarlos correctamente en la plantilla **cuenta.html**. Se convierte su valor a decimales y posteriormente se obtiene la Transaction asociada a ese Split mediante una consulta. Mediante el GUID de la Transaction se obtiene el Split complementario al actual Split, que se encontrará en la cuenta destino, de la que se recupera el nombre para mostrarlo en plantilla. Se calcula el balance que había en el momento en el que se produjo ese movimiento sumando o restando según el tipo de cuenta.

Se crea una tupla en la que se introduce el Split, la transacción, el nombre de la cuenta destino, el balance y el valor. Estos son los datos necesarios para mostrar en la plantilla los detalles de los movimientos. Por cada Split se guarda la tupla creada en una lista de asientos, que será la enviada, mediante el método `render_to_response`, a la plantilla **cuenta.html**.

Ya en la plantilla y utilizando esta lista, se recorre y se muestra cada movimiento con sus datos correspondientes, es decir: Fecha, Número, Descripción, cuenta destino, estado de reconciliación, valor del movimiento y balance de la cuenta en el instante posterior a producirse el movimiento. Todo ello en el formato original mostrado por GNUCash.

4.3.d) Borrar entrada

Partiendo del GUID del Transaction, se obtienen los dos Splits asociados y, tras mostrar al usuario una página de confirmación de borrado, se eliminan los 3 campos de la base de datos. Para determinar si se ha mostrado el aviso al usuario se utiliza el parámetro “confirmación” recibido por GET. Si no existe, el usuario viene de pulsar la papelera, por tanto se muestra el aviso de la platilla **borrar.html**, si existe, viene del aviso, por tanto se elimina (si procede) y se devuelve a la vista de la cuenta. En caso de que se entre por error, sin indicar un GUID de transacción, se mostrará la pantalla principal.

Sección 4.4 Tratamiento de los datos

Durante el desarrollo del proyecto ha sido necesario tratar datos para que los reconociese GNUCash, para convertirlos en las estructuras de datos que se necesitasen en cada momento o para poder mostrarlos en las plantillas de la forma adecuada. Esto ha requerido de búsqueda de documentación y de mucho tiempo y pruebas para conseguir trasformar y adaptar los datos a las necesidades y particularidades de la aplicación. El hecho de haber utilizado Python ha sido de gran ayuda ya que proporciona un gran número de funciones útiles para esta tarea.

4.4.a) GUID

GNUcash utiliza una variación de los UUID como identificadores únicos en muchas de sus tablas, lo que nos llevó a buscar una forma de conseguir crear estos identificadores de forma automática, necesarios por ejemplo para identificar Splits y transacciones y, en general, cualquier entrada de cualquiera de las tablas de la base de datos, ya que los utiliza como claves primarias. Los GUID de GNUCash son cadenas alfanuméricas de 32 caracteres, sin separación entre ellos, almacenado con el tipo **varchar**.

El proceso de generación de los GUID consistió primero en importar el módulo de Python y generar un objeto uuid4 de la forma:

```
>>> uuid.uuid4()
    UUID('16fd2706-8baf-433b-82eb-8c7fada847da')
```

Luego con la función **str** de Python convertimos el objeto uuid en un String que se pueda modificar. Ya como String, con la función **replace** se eliminan los guiones, remplazándolos por “” (cadena vacía). La función final para generar los GUID queda de la siguiente manera:

```
guid = str(uuid.uuid4()).replace('-','')
```

Lo que genera Strings que al insertarlos en base de datos no generan incompatibilidades con GNUCash, y que cuyo formato es:

```
'16fd27068baf433b82eb8c7fada847da'
```

4.4.b) Consultas a la base de datos

En todas las vistas se realizan consultas constantes a la base de datos a través de los modelos. Este proceso en Django es muy simple ya que sólo hay que importar los modelos en las vistas y hacer las consultas de las siguientes formas:

Si por ejemplo hay que extraer una cuenta con todos sus datos mediante el GUID de esa cuenta, el proceso sería aplicar al modelo **Accounts** la función **get()** pasándole como argumento el GUID. El fragmento de código sería:

```
account = Accounts.objects.get(guid=mi_guid)
```

Teniendo la cuenta se podría acceder al valor de cualquier campo de la cuenta con un punto y el nombre del campo. Por ejemplo el tipo de cuenta:

```
tipoCuenta = account.account_type
```

Para extraer todas las tuplas de una tabla filtradas por una cláusula, como por ejemplo extraer todos los Split de una cuenta concreta, se hace mediante el método **filter()**, pasándole como argumento el GUID de la cuenta:

```
splits = Splits.objects.filter(account_guid=mi_guid)
```

Es importante explicar cómo funcionan estas consultas ya que se utilizan recurrentemente en las vistas de la aplicación.

4.4.c) Tipos de cuentas GNUCash

- ASSET: Activo, cosas con lo que se puede pagar. Generalmente marcador de posición.
- BANK: Cuenta bancaria. Puede tener intereses.
- CASH: Efectivo. Forma parte del activo.
- CREDIT: Tarjeta de crédito.
- EQUITY: Equidad del patrimonio. Para cuentas de apertura o cierre de ejercicio.
- EXPENSE: Gastos.
- INCOME: Ingresos.
- LIABILITY: Responsabilidad, pasivo, deudas.
- ROOT: Cuenta raíz. Padre.

4.4.d) Listar cuentas

La obtención de una lista de cuentas que mostrase sólo lo que interesaba en la plantilla **lista_cuentas.html** requirió de un proceso de tratamiento de datos complejo que se detallar a continuación.

A partir de la cuenta raíz del libro de cuentas, en la vista `listaCuentas` se llama a una función auxiliar llamada `lista_cuentas_rec`, que saca las cuentas hijas de esa cuenta y las añade a una lista en la que se forma una tupla con el nombre completo de esa cuenta (incluyendo la ruta de cuentas padre) por un lado, y su GUID por otro. Por cada hijo obtenido se llama recursivamente a esta función, de modo que en la lista de cuentas añade las hijas concatenando el nombre de la cuenta actual con sus padres y su GUID, de forma que quedaría algo como lo siguiente, poniendo como ejemplo tres cuentas en las que cada una es hija de la anterior:

```
accounts = [(Activo,guid1),
(Activo:Activo circulante,guid2),
(Activo:Activo circulante:Metálico,guid3)]
```

Ordenamos la lista de cuentas para recorrerla y por cada cuenta separar por dos puntos los nombres concatenados, añadiéndolos a otra lista y sobrescribiendo la primera posición en la que estaban los nombres concatenados por la nueva lista. Por cada cuenta ejecutamos el siguiente algoritmo, utilizando la función Split de Python para separar por ":" la cadena:

```
account_list[i] =
account_list[i][0].split(':')[1:], account_list[i][1]
```

Es decir, si hay una cuenta en la que están concatenadas sus cuentas padres en la posición cero (*Activo:Activo circulante:Metálico*), se crea una lista nueva en la que en la posición cero tendríamos "*Activo*", en la uno "*Activo circulante*", y en la dos "*Metálico*". Esta nueva lista se inserta en la posición de dicha cuenta dentro de la lista `accounts`. Por lo que en esta lista `accounts`, la cuenta "Metálico" tiene en la posición cero esta nueva lista (*Activo,Activo circulante,Metálico*) y en la posición uno el GUID de "*Metálico*", quedando de la siguiente forma:

```
accounts = [((Activo),guid1),
((Activo,Activo circulante),guid2),
((Activo,Activo circulante,Metálico),guid3)]
```

Se utiliza una lista auxiliar `Account_toHTML`, que reorriendo la lista de cuentas, por cada una va añadiendo a `Account_toHTML` el nombre de la cuenta (el último nombre de la lista) concatenado a un número de espacios multiplicado por cada padre que tenga, para que al mostrarlo en la plantilla como HTML salga tabulado como un árbol de directorios, junto con una etiqueta **href** con el GUID de la cuenta, y el total, calculado mediante una función auxiliar `totalizarCuenta`.

```
account_toHTML.append(('     ' *
(len(account_list[i][0])-1) + '<a href="/cuenta.html?guid='
+ account_list[i][1] + '">' + account_list[i][0][-1] +
'</a><br/>',totalizarCuenta(account_list[i][1])))
```

Esta lista se envía a la plantilla **lista_cuentas.html** para que muestre el directorio de cuentas con enlaces a cada cuenta.

4.4.e) Totalizar cuenta

Para calcular el total de cada cuenta, es decir la suma de los movimientos de esa cuenta y la de sus hijos, se llama desde `listaCuentas` a una función `totalizarCuenta` a la que se le pasa el GUID de la cuenta.

En dicha función se obtienen los Split asociados a esa cuenta y se suma su valor a una variable `amount`. En el caso de que las cuentas sean de tipo "**income**", "**liability**" o "**equity**", no se suma sino que se resta el valor del Split.

Luego se obtienen los hijos de esa cuenta y se recorren llamando recursivamente a `totalizarCuenta` pasándole el GUID de cada hijo, y añadiendo a `amount` los totales de cada cuenta.

4.4.f) Ver cuentas

Para mostrar los movimientos de cada cuenta concreta en la plantilla **cuenta.html**, desde la vista verCuentayAsiento hay que realizar un filtrado de los datos para tratarlos y mostrarlos de la forma en la que lo hace GNUCash.

Lo primero que se hace en esta vista, es extraer los Split de la cuenta deseada, cuyo GUID llega por GET (o POST si se ha insertado un asiento). Se hace de la siguiente forma:

```
splits = Splits.objects.filter(account_guid=request.GET['guid'])
```

Recorriendo la lista de Splits, por cada uno se realizan una serie de operaciones. Se convierte su valor a decimales, ya que por ejemplo 10,50€ en base de datos se guardará como 1050:

```
split.value_num = split.value_num/float(split.value_denom)
```

Mediante consultas, se obtiene la transacción asociada al Split y a partir de ésta el Split complementario, desde el que se recuperará la cuenta a la que va el dinero de ese movimiento consultando a Accounts con el GUID de dicho Split.

Por cada Split se genera una tupla con las fechas (para ordenar), el Split, la transacción, el balance y el valor, que se añade a una lista de asientos:

```
t=(transaction_split.post_date,
transaction_split.enter_date,split,transaction_split,
split2_account,balance,-split.value_num)
asientos.append(t)
```

Se ordenan los asientos por fecha, con la función sort() ya que la fecha es el primer valor de la tupla. Después se va sumando el valor de cada Split al balance de la cuenta, salvo en el caso de que las cuentas sean de tipo "**income**", "**liability**" o "**equity**", que no suman sino que restan el valor del Split.

Por último se actualiza la tupla que se pasará a la vista, quitando las fechas y guardando el balance asociado a cada asiento. Los elementos se van insertando en la tupla con el método append(asiento), pero podría utilizarse otro método para insertar al principio e invertir el orden en que se muestran los movimientos.

Esta vista integra también el formulario para añadir asientos, por lo que se comprueba si se ha recibido una llamada por el método POST. En caso afirmativo, se realizan las operaciones necesarias para tratar el formulario recibido e insertarlo en la base de datos.

Se crea una variable de tipo Transactions y dos de tipos Splits, en ellas se genera un GUID para cada una y se van introduciendo los datos correspondientes según los recibidos por POST. En uno de los Split, el campos `account_guid` corresponde al GUID recibido por GET, de la cuenta que se está mostrando.

Una vez asignados todos los valores se guardan en la base de datos con la función `save()` y se actualiza la vista de la cuenta.

Los formularios vienen de los definidos en forms.py, que ya se han explicado anteriormente, y se pasan a la vista a través del método render.

Sección 4.5 Dificultades encontradas durante la implementación

El desarrollo de un proyecto basado en una aplicación existente siempre supone un reto en el cual pueden surgir complicaciones inesperadas. La aplicación existente puede tener sus peculiaridades y éstas hacer que adaptar otra versión sea algo laborioso o que produzca dolores de cabeza y atascos en la implementación del proyecto.

4.5.a) Aprendizaje de las tecnologías

Cabe destacar que ninguno de los componentes del grupo tenía conocimientos de las tecnologías que se usan para el desarrollo de la aplicación, tanto Python como Django, lo que supuso un problema desde el principio, obligando a dedicar más tiempo al estudio y aprendizaje de las tecnologías que al desarrollo en sí de la aplicación. Aprender cómo usar el framework Django requirió del seguimiento de muchos tutoriales online y de sobretodo la implementación como aprendizaje de pequeños proyectos Django enfocados a la integración de formularios con una base de datos.

4.5.b) GUID

Ya se ha explicado anteriormente como obtener el GUID utilizado como clave primaria en las entradas de tabla de la base de datos de GNUCash, por lo que no volveremos a detallarlo. Pero cabe destacar la dificultad que supuso este hecho. Los modelos de Django crean por defecto un id que se utiliza como clave primaria y es autoincremental. Esto modificaría también la base de datos y no sería compatible con GNUCash.

El formato en que se guardaba el dato no era estándar, por lo que fue necesario aplicar un tratamiento al UUID obtenido. Las primeras pruebas, con inserciones a mano en la base de datos, con estos UUID obtenidos, daban un error al abrir la base de datos con GNUCash, que avisaba de que la base de datos podía haber sido modificada por una aplicación externa, como indicando que algo podría estar yendo mal y el funcionamiento de la aplicación verse afectado negativamente por este hecho.

Existe en el repositorio SVN de GNUCash unos ficheros referentes al GUID, incluso alguno en Python, pero no ha sido posible hacer uso de ellos al no poderse integrar fácilmente en nuestra aplicación.

4.5.c) Renombramiento de las tablas

La base de datos de GNUCash es bastante extensa, como se pudo ver en el diagrama entidad-relación capítulos antes. Aunque para esta aplicación no se utilicen todas las tablas, se consideró oportuno hacer la migración de todas ellas para favorecer el desarrollo de nuevas funciones en el futuro.

Hacer eso de forma manual habría sido demasiado laborioso y propenso a errores humanos. Como se ha explicado anteriormente, Django proporciona una facilidad en este aspecto. El problema vino cuando al usar ese comando, creaba los modelos, sí, (aunque era pertinente realizar algunas comprobaciones) pero también hacía modificaciones en la base de datos. Renombraba todas las tablas, concatenando a su nombre original (el utilizado por GNUCash) con el prefijo del nombre de la aplicación Django. Quedando por ejemplo la tabla **Accounts** de la forma **tfgapp_Accounts**. Esto evidentemente no iba a ser entendido por GNUCash en el escritorio.

Fue necesario buscar otras alternativas para evitar que esa concatenación se realizase. En concreto, se especificó el nombre en el campos `db_table` de la clase `Meta` de cada uno de los modelos autogenerados.

4.5.d) Cálculo del balance de una cuenta

Es curioso que en ningún lado se guarda el balance de la cuenta tras un movimiento ni el balance total actual de la cuenta. Puede tener sentido para permitir meter asientos nuevos entre otros ya existentes (habría entonces que modificar los balances de todos los asientos con fecha posterior). Además así se permite mostrar el balance resultante tras cada asiento por órdenes distintos, por ejemplo, por fecha valor o por fecha de introducción. Más fácil sería almacenar un total como balance actual de una cuenta, en una variable más de Accounts. Pero GNUCash no lo tiene así configurado.

Por un lado, al mostrar la lista completa de las cuentas, se debe mostrar el saldo de cada una de ellas. Es necesario por lo tanto recuperar todos los asientos (*Splits*) de esa cuentas y hacer la suma de todos sus importes.

Por otro lado, al acceder a una cuenta para ver sus movimientos, tras cada uno de ellos se debe mostrar también el balance resultante tras ese movimiento. El hecho de que las plantillas en Django no permitan la modificación de variables, ni apenas operaciones desde el propio *template*, supuso un problema debido a que desde la vista se le pasa a la plantilla una lista de asientos, en la que se incluye cada *Split* asociado, con el valor del movimiento. Este valor es el que se almacena en base de datos, es decir, puede ser positivo o negativo, pero en la plantilla siempre se muestra como positivo, variando su posición entre la columna del "debe" y del "haber". A su vez, se debía hacer un cálculo del balance resultante tras introducir el asiento.

Tanto este cálculo como la modificación de mostrar el valor siempre en positivo hubiese sido muy simple hacerlo directamente en la plantilla a medida que va mostrando cada movimiento, simplemente cambiándole el signo al valor y sumándolo o restándolo a una variable balance.

Al encontrar el problema de no poder hacer cálculos en la plantilla, se modificó el comportamiento de la vista para enviar a la plantilla los datos ya tratados.

En la vista se creó una variable `balance` que se incrementa dependiendo del tipo de cuenta y del valor del movimiento. Se pasa a la plantilla este valor también con el signo cambiado. Asociando estos datos en una misma tupla del movimiento correspondiente. Al pasarle una lista con estas tuplas a la plantilla se pueden mostrar los datos de la forma que lo hace GNUCash.

4.5.e) Integración del formulario en la vista de cuenta

Django facilita tanto la creación de formularios que genera automáticamente el código HTML para crearlo y mostrar los campos necesarios. Permite elegir distintos formatos, como párrafo, como tabla... pero ninguno se adaptaba bien a la tabla en la que debía mostrarse, además de que no se debían mostrar también las etiquetas, puesto que estas están en la primera fila de la tabla. Para ello se recorren desde la plantilla todos los campos de cada formulario y se inserta únicamente el *field* de cada campo, entre las etiquetas de columnas (`<td>`).

Por otro lado, el ancho de estos campos era automático y descuadraba ligeramente la vista final de la página HTML, por lo que se añadió al **CSS** la condición de que éstos campos ocupasen un porcentaje de ancho de la columna en vez de un ancho fijo.

Para mostrar una cuenta concreta, se la identificaba pasando el GUID por GET. Al querer integrar el formulario en la misma plantilla, resultó que éste pasaba datos por POST y no se podía interpretar también el GET. Esto se solucionó poniendo un campo de tipo *hidden* (oculto) en el formulario, de forma que en caso de recibirse un nuevo asiento, identifique la cuenta que debe mostrar por el GUID recibido desde POST en vez de desde GET.

Capítulo 5 Resultados visibles

La aplicación presenta en su vista inicial una lista de las cuentas disponibles y el balance de cada una de ellas. Esta vista es similar a la mostrada por la pantalla principal de la aplicación de escritorio GNUCash, mostrando también las cuentas en cascada en función de la categorización existente.

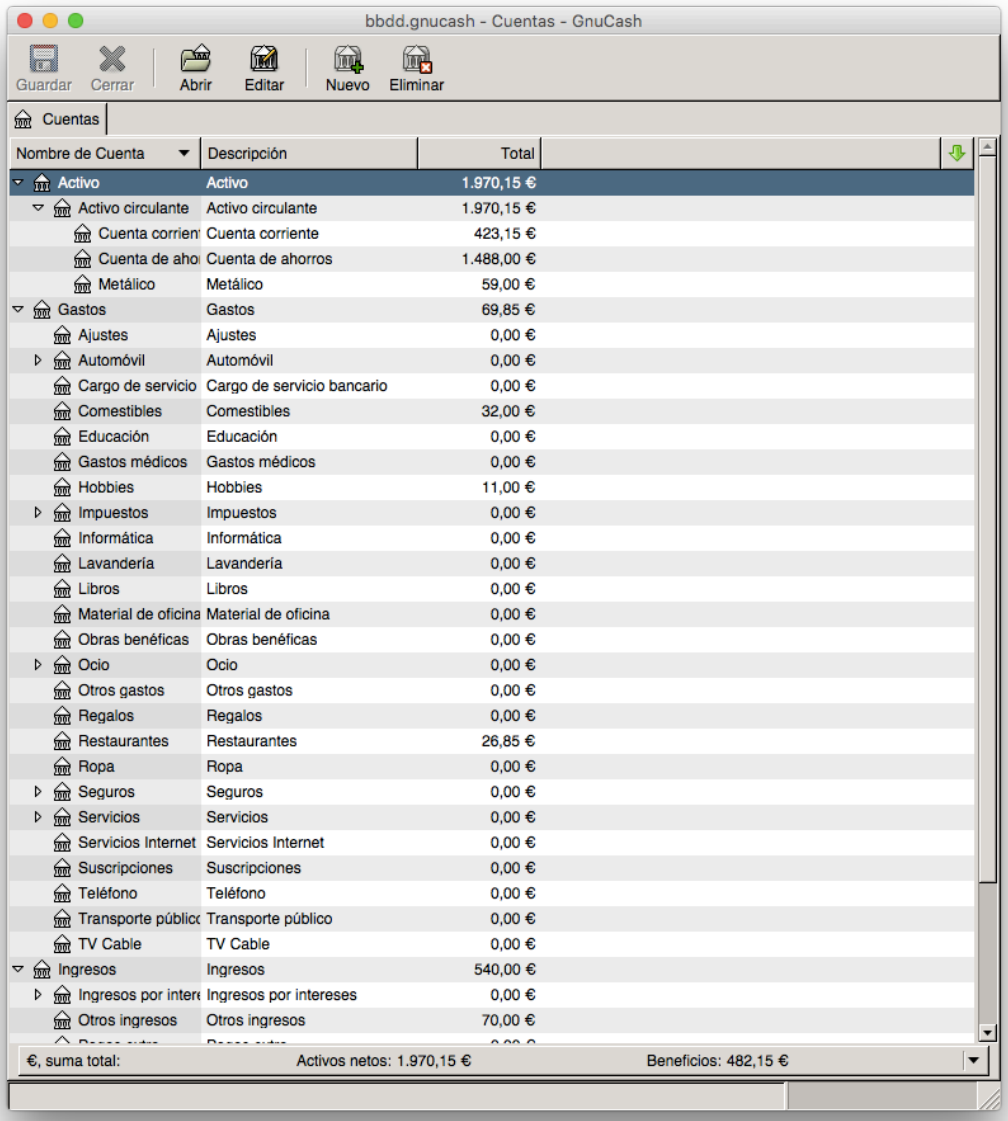

Figura 10. Vista principal de la aplicación de escritorio GnuCash

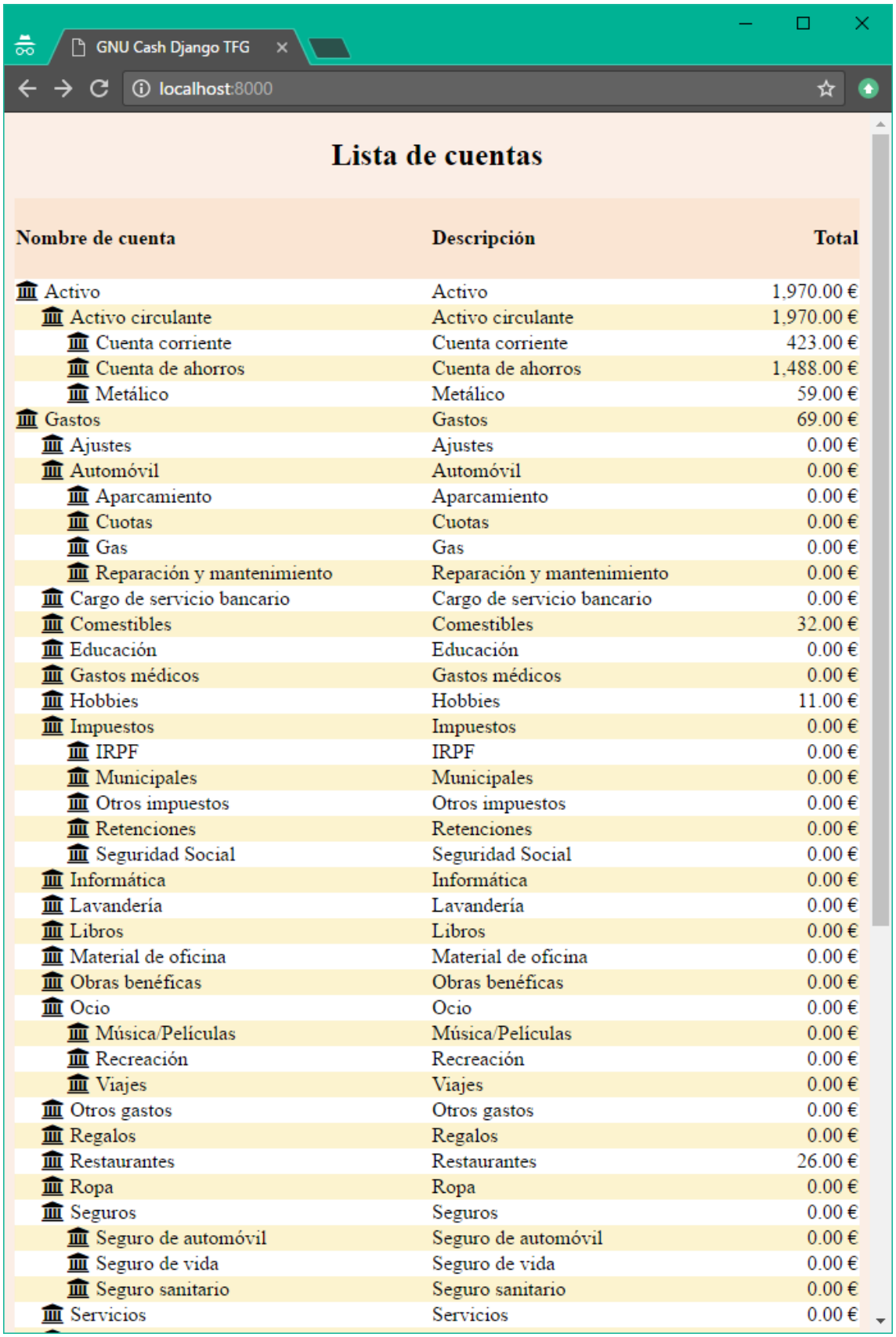

Figura 11. Vista principal de la aplicación desarrollada

Al pinchar en cualquiera de las cuentas se accede al detalle de la misma. En esta página se muestran los movimientos en detalle, ordenados del más antiguo al más reciente. Los detalles incluyen una fecha operación y fecha valor, descripción, cuenta destino, importe de entrada o de salida y balance posterior.

La última línea de la lista de movimientos detalle de una cuenta está compuesta por campos vacíos de un formulario en el que rellenar los datos correspondientes para crear un nuevo movimiento. Al enviar el formulario, si los datos son válidos, se inserta automáticamente el asiento y se actualiza la vista de la cuenta en consecuencia.

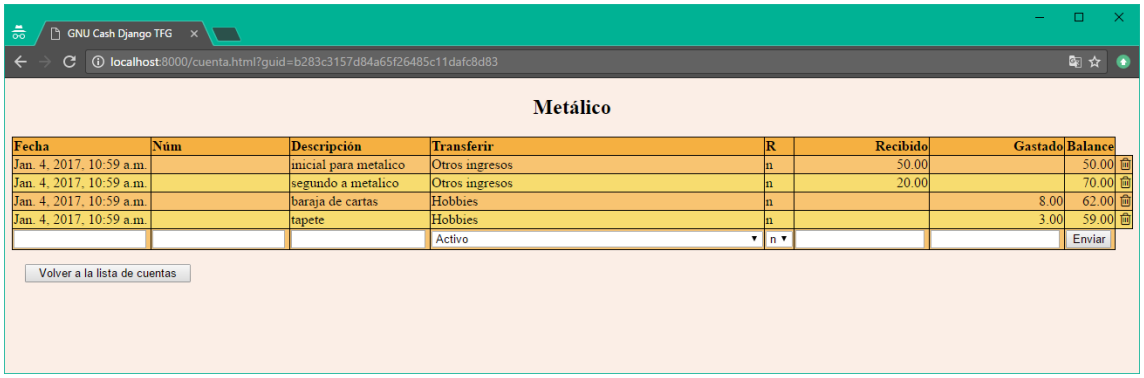

Figura 12. Vista detalle de cuenta en la aplicación

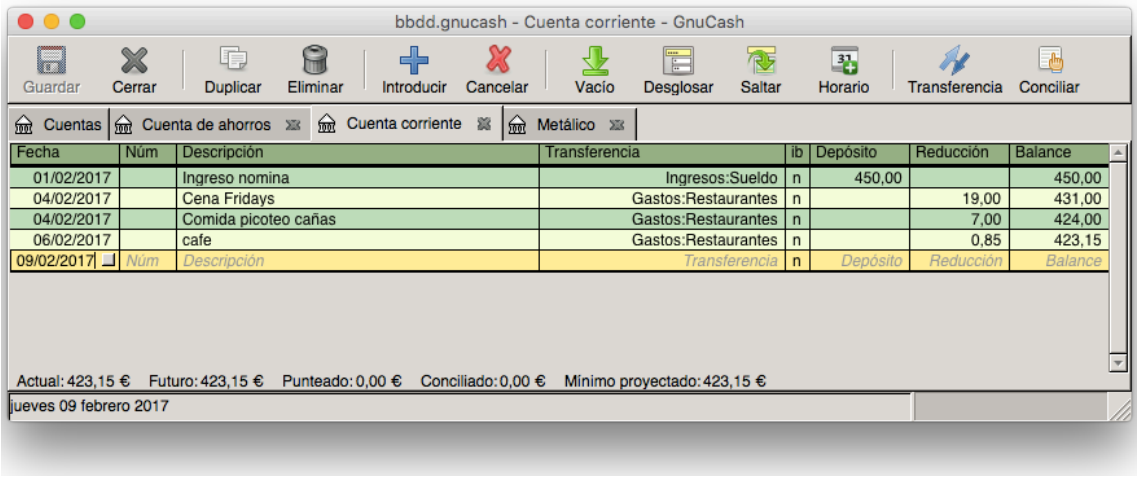

Figura 13. Vista detalle de cuenta en GnuCash

Para crear un asiento nuevo el formulario dispone de un desplegable que muestra todas las cuentas con su ruta completa como se ha explicado en esta memoria y se muestra en esta figura:

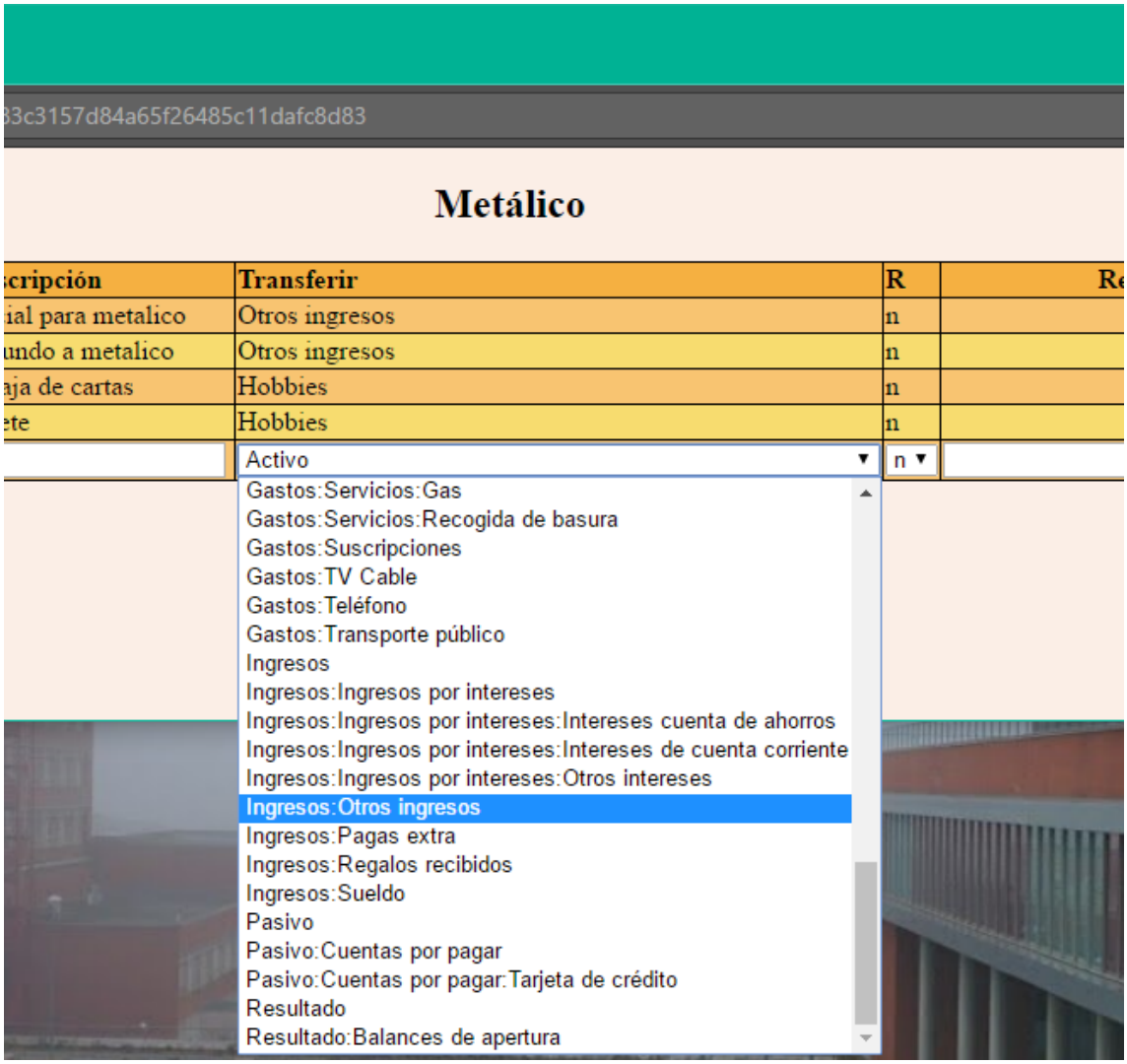

Figura 14. Desplegable con lista de cuentas

En caso de pulsar el botón de borrado se pide confirmación al usuario:

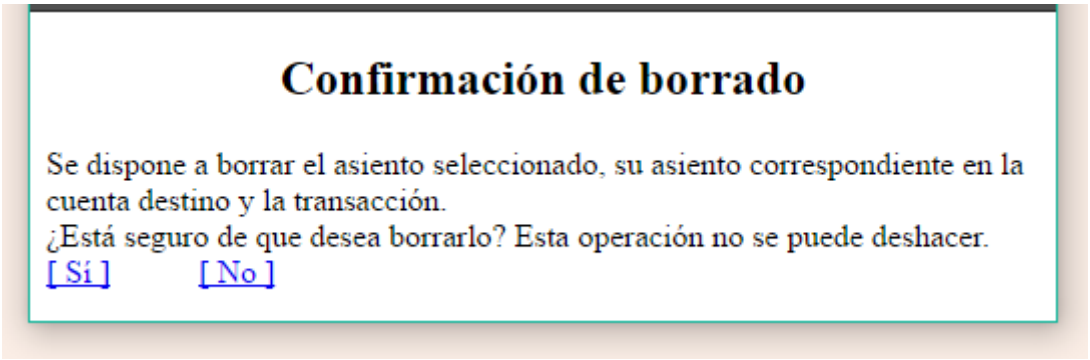

Figura 15. Confirmación de borrado

Por último, existe una página separada que consta únicamente del formulario para crear un nuevo movimiento. A diferencia del anterior, en este se pide al usuario la cuenta origen y la cuenta destino, como es lógico, al no estar integrado este formulario dentro de una cuenta. Esta página fue una de las primeras en crearse, a priori como prueba, pero se ha decidido mantener pese a estar fuera del núcleo de la aplicación por conformar una utilidad que va muy en la línea de uno de los objetivos del proyecto: Poder hacer anotaciones contables en las cuentas de GNUCash de manera rápida y desde cualquier dispositivo.

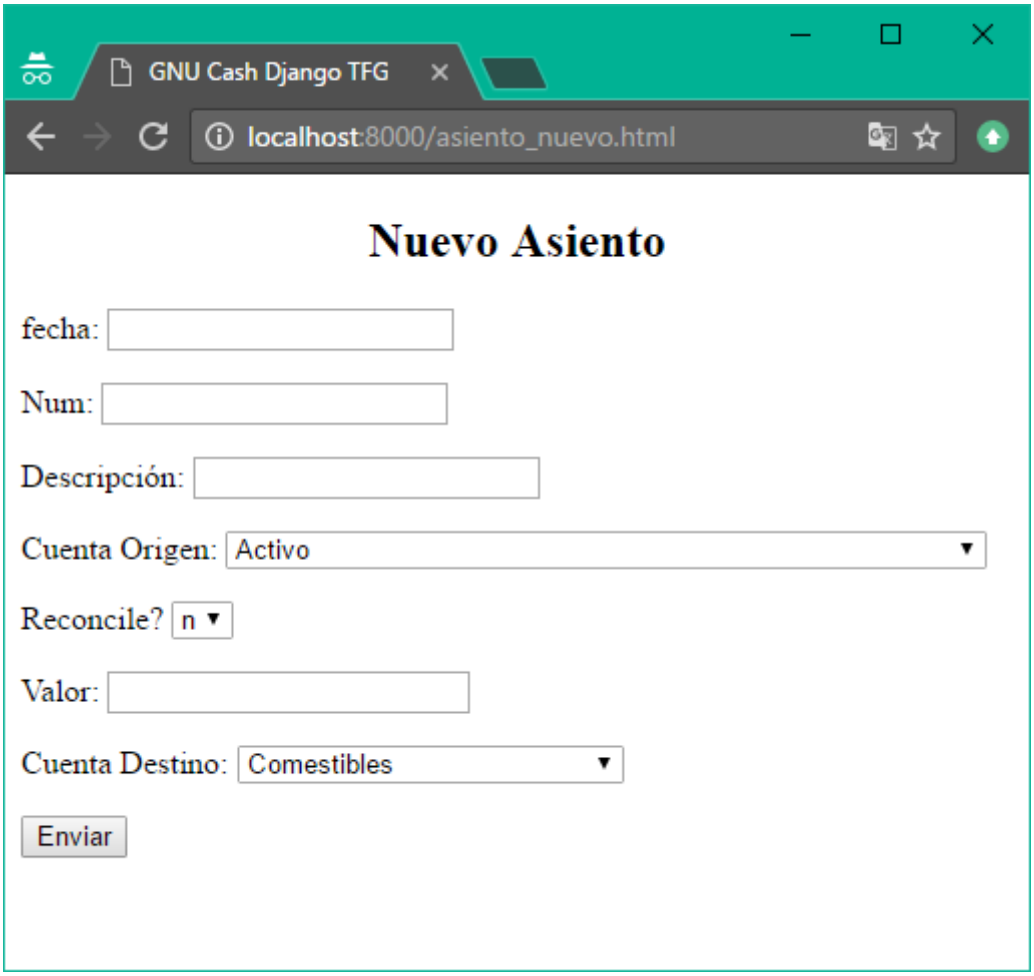

Figura 16. Formulario de inserción de asiento

Capítulo 6 Conclusiones y trabajo futuro

Se ha conseguido desarrollar una aplicación que cumple con los objetivos básicos que se buscaban al plantear el proyecto. En muchas ocasiones se buscó algo más ambicioso que permitiese utilizar algunas más de las múltiples posibilidades que ofrece GNUCash, y eso es técnicamente posible en la mayoría de los casos.

El desarrollo del proyecto se ha demorado más de lo esperado por la necesidad de habituarse a un lenguaje y framework con los que no habíamos trabajado anteriormente y la de leer documentación sobre la aplicación en la cual nos basamos así como investigar su funcionamiento. Por este motivo se han dejado por el camino algunas opciones que queríamos añadir y finalmente no ha sido posible.

La complejidad de la aplicación ha supuesto un hándicap. Trabajar basándose en algo que ya existe siempre es complicado, por más que exista mucha documentación o comunidad de desarrolladores detrás. Más teniendo en cuenta la cantidad de funciones accesorias, o al menos irrelevantes para nuestro proyecto, que contiene, que hace necesario un filtrado en el periodo de aprendizaje y habituación.

En conclusión, este proyecto ha supuesto un desafío que finalmente hemos conseguido superar. Las principales funciones que se esperaban han sido cubiertas y se ha logrado la integración con la aplicación de escritorio. Así y todo, existen más funciones que sería interesante desarrollar pero que finalmente no se han implementado.

Por un lado, sería interesante dotar a la aplicación de un aspecto más limpio e intuitivo para el usuario. Sustituir los actuales **select** con las cuentas por campos de texto que se completen automáticamente buscando coincidencias en los nombres de las cuentas según el usuario va escribiendo. Y otro tipo de aspectos gráficos de la Interfaz de Usuario que habría que pulir.

Por otro lado, sería también bastante interesante poder acceder a algunas de las herramientas que ofrece GNUCash en su versión de escritorio. Por ejemplo, la generación de informes y de gráficas, que se generan precisamente en una página HTML. Es algo que se ha planteado y que se considera factible, pero que no se ha implementado por falta de tiempo.

La aplicación presenta ahora algunas vulnerabilidades de seguridad. Principalmente, los datos no viajan cifrados por la red al no correr en un servidor HTTPS. Esta sería la primera mejora importante que habría que adoptar antes de lanzar la aplicación, sobre todo teniendo en cuenta que maneja datos económicos, que son potencialmente sensibles.

Además, la página del servicio no pide ningún tipo de identificación ni login. Cualquier persona que conozca la URL puede acceder a la aplicación, a sus datos y modificar o añadir lo que desee. Habría que añadir un control de acceso por cuentas de usuario, cookies o sesiones. Esto va relacionado con otra característica que quisiéramos añadir, que es la de permitir soporte multiusuario.

Añadir una página de identificación en la que el usuario inicie sesión. De esta forma la aplicación podría servir para llevar distintas bases de datos de diferentes usuarios. Sería necesario programar un mecanismo mediante el que, una vez conocidos los datos del usuario, se configure la aplicación (en settings.py) para que acceda a la base de datos del usuario que se ha identificado.

De aplicarse la opción multiusuario, sería necesario que cada usuario tuviera no sólo una base de datos individual, para que fuera compatible con GNUCash, sino también un usuario diferenciado de acceso a la base de datos, con permisos única y exclusivamente sobre su base de datos.

Chapter 6 Conclusions and future work

An application that fulfils the basic initial objectives of the project has been developed. Another features present in GNUCash have been investigated, and all of them are technically feasible.

The development of the project has been delayed more than expected manly because the need of familiarizing a new language and a new framework that we have not work before with. We also needed to read the documentation and investigate about already existing application that is the base of ours. This is the main reason of discarding some options that were initially planned.

The complexity of GNUCash has been a handicap. Try to work over an preexisting application is always complication even if there is much documentation and a developer community behind it. Moreover there is a huge number of functions that are less relevant to our project. This makes an big effort to filter information in the learning period.

In conclusion, this project has been a challenge that have been successfully achieved. The main functions have been covered and the integration with the desktop application has been achieved. Even though, there is a lot of functionality that have not been implemented.

On the one hand, it would be interesting to provide a cleaner and intuitive aspect for the user. The substitution of the current "select" with the accounts with auto-completing text fields is a must work for the immediate future. There are another aspects of the User Graphic Interface that should be improved.

On the other hand, it would be interesting to access to some tools that GNUCash offers in the desktop version. For instance, the generation report and graphs that actually are presented in an HTML page. This seems quite feasible but there has not been much time to research about it.

Currently, the application has security vulnerabilities. Mainly the data are not encrypted because the protocol HTTPS is not used. This is the first feature that should be implemented before launching the application, specially taking into account it manages sensitive economic data.

Also, the web application do not have any authentication mechanism. Anybody knowing the URL has access to the application and its data and make any changes. It is also necessary to include a an access control to users' accounts to provide the application with multi-user support. It should be also necessary that each user should have its own GNUCash database with permissions to access its own database.

Incorporate an identification web to login. That way, the application could be used to manage different data bases of different users. It would be necessary program a device in which one, once known the users data, it configures an application (settings.py) to access in the identified user database.

Contribución de los participantes

Carlos Martínez Sacristán

Respecto a la aportación de cada alumno al proyecto, cabe destacar que no ha habido un reparto de tareas estricto, por lo que ambos generalmente hemos trabajado en las mismas materias conjuntamente, aunque hay tareas y desarrollos en los que un componente ha trabajado de una forma más destacada que el otro.

De forma individual, lo primero a hacer fue instalar el programa GNUCash y comenzar a usarlo en el día a día para aprender su funcionamiento básico y centrarnos en qué queríamos desarrollar en nuestra aplicación web. Tras tener creado un entorno en una máquina virtual en el que poder desarrollar, me centré en buscar documentación de Django y seguir tutoriales para poder conectar un proyecto Django con una base de datos en MySQL, creando varios proyectos de prueba, en los que poco a poco se fueron integrando nuevas funcionalidades, creando formularios que modificaban la base de datos, y aprendiendo la forma en la que se relacionan las vistas con las plantillas en Django. También se hicieron pruebas de sincronización de los modelos con las tablas de la base de datos para ver de qué forma podíamos integrar la base de datos de GNUCash en nuestro proyecto.

Durante el desarrollo, hubo mucha tarea de investigación sobre el funcionamiento de GNUCash y la forma en que podíamos desarrollarlo en Django y Python. Primero, con las relaciones entre las tablas de la base de datos, en las que creamos asientos en el programa original y posteriormente vimos el comportamiento de la base de datos desde phpmyadmin, para así poder investigar cómo desarrollarlo en Django, y obtener finalmente el mismo funcionamiento en nuestro proyecto que en la aplicación original.

También requirió de investigación la forma de tratar los datos en Python, el funcionamiento de las colecciones, la búsqueda de métodos apropiados para cada momento, ya que a medida que se desarrolla la aplicación surgen requisitos en cuanto a la forma de obtener y mostrar los datos que requieren de modificaciones con Python, y al ser un lenguaje con el que no habíamos trabajado, tuvimos que buscar constantemente documentación.

Hubo que crear y configurar el proyecto que posteriormente usaríamos como proyecto definitivo, realizando la configuración para poderlo usar con MySQL y preparándolo para poder integrar la base de datos de GNUCash.

Aunque la mayor parte del proyecto está implementado de forma conjunta, hay funcionalidades desarrolladas por mi parte, como la parte de la lógica e implementación de mostrar los detalles de cada cuenta. Para ello hubo que investigar tanto la forma en la que mostraba los datos GNUCash como la forma de tratar los datos para mostrarlos de la misma manera en nuestro proyecto.

Otra tarea fue buscar documentación para crear un log de errores que nos permitiese debuguear y encontrar fallos durante el desarrollo, y que posteriormente hubo que implementar.

Para que la aplicación tuviese un aspecto similar al de GNUCash, se fueron aplicando estilos con CSS a las plantillas de la lista de cuentas y de ver cada cuenta, de forma que ambas plantillas quedaran con un formato de tabla bastante intuitivo a la hora de usar la aplicación, y lo más parecido posible al programa ya desarrollado.

La aplicación ha requerido de pruebas constantes durante su desarrollo que nos han permitido ver fallos en la forma de mostrar los datos o errores que podían afectar al correcto funcionamiento de la base de datos. Hemos realizado modificaciones a raíz de estas pruebas solucionando problemas que no eran graves pero que si provocaban errores tanto en la forma de visualizar los datos como en cálculos con éstos que podían guardarse, por lo tanto mal, en la base de datos. Fruto de esto hemos arreglado cosas como el nombre de las columnas dependiendo de la cuenta que se esté mostrando (haciéndolo de acuerdo a GNUCash), el formato de los decimales tanto en la lista como en los detalles de la cuenta, el tener en cuenta el tipo de cuenta para calcular el balance, debido a que unas suman al balance final y otras restan, o la ordenación de la lista para que muestre las cuentas sólo con su nombre sin el nombre de sus padres delante, entre otras.

Enrique Ugedo Egido

La elaboración de este proyecto ha sido bastante horizontal y ambos componentes hemos trabajado en la mayor parte del mismo de forma paralela y ayudándonos mutuamente, sin haber existido una separación estricta de funciones. Dicho esto, es evidente que ambos hemos desarrollado diferentes partes necesarias para el proyecto de forma individualizada.

Lo primero de todo fue crear el entorno de pruebas en el que íbamos a familiarizarnos con el funcionamiento de la aplicación GNUCash, con lenguaje de programación Python y empezar a usar Django para desarrollar aplicaciones web de prueba. Para ello se creó la máquina virtual en VirtualBox con Ubuntu e instalaron los paquetes necesarios para la integración de todos los servicios, así como la aplicación de escritorio de GNUCash y los complementos que permiten su uso con bases de datos MySQL.

Después fue necesario realizar un trabajo de investigación sobre GNUCash por lo que me encargué de buscar toda la información relativa a la organización de la base de datos, así como su diagrama entidad-relación y las peculiaridades de sus campos clave (*GUID*) y la forma de obtenerlos mediante comandos de Python y de integrar que éste se incluyera de forma automática al rellenar un formulario. También se estudió el comportamiento de los distintos tipos de cuenta para poder totalizar su balance en la función correspondiente.

Para continuar con esta investigación se consultaron los archivos de código fuente, alojados en un svn, en busca de funciones que nos pudieran ser útiles para nuestra aplicación, así como la documentación correspondiente. También me suscribí a las listas de correo de desarrolladores para consultar algunas dudas y para estar al tanto de las novedades o bugs que se encontraban, por si alguno de estos pudiera afectar al normal desarrollo de nuestro proyecto. Del mismo modo, se utilizaron en alguna ocasión la sala de chat de desarrolladores en IRC para preguntar alguna duda concreta.

Como el desarrollo de la aplicación se ha llevado a cabo directamente en el servidor, se creó una carpeta de Dropbox, con una cuenta específica, que permitía la sincronización de todos los ficheros. El motivo principal era poder utilizar entornos de programación más cómodos que la línea de comandos.

Para poder colaborar también en la parte de programación propiamente dicha seguí los tutoriales que me recomendó Carlos para aprender a hacer formularios en Django y alguno más que encontré, aparte de hacer pruebas propias de mi curiosidad para comprobar el comportamiento del programa con cosas poco habituales.

Desarrollé algunas funciones de especial interés para el proyecto como la que obtiene el balance total de la cuenta, la lista de cuentas jerarquizada por categorías (o cuentas padre), una gran parte del método que recupera los datos de un formulario para grabarlos en distintas tablas de la base de datos y la función que permite borrar entradas desde la vista de cuenta eliminando tanto el *Transaction* como ambos *Splits*.

Tal como se desarrolló el formulario individualizado de añadir un asiento nuevo. También realicé la integración del formulario de asiento nuevo en la vista detalle de una cuenta. Cambiando algunos aspectos como que se muestre de forma ordenada por fecha, lo que obligó a hacer un nuevo mecanismo de cálculo del balance tras cada asiento, que también adapté.

Por último, me he encargado de realizar pruebas de funcionamiento durante todo el proceso y especialmente al final. Detectando algunos fallos como la posibilidad de que al crear un asiento nuevo se pueda producir un fallo de algún tipo al escribir el *Split* en la base de datos, pero posterior a la escritura del *Transaction*, lo que crearía una inconsistencia en los datos al existir una transacción sin asientos asociados, o con solo un asiento. Este mismo fallo se da también para el caso de eliminar.

Consecuencia de estas pruebas se han realizado algunas modificaciones para ajustar el funcionamiento de la aplicación al esperado, por ejemplo: Aplicar la diferenciación antes descrita de tipos de cuentas a la hora de calcular su balance (las cuentas de ingresos, deudas y equidad suman los asientos negativos, por así decirlo) y dar formato a la presentación de los balances, para mostrar 2 decimales y alineados a la derecha.

Referencias

[1] Django Project, “Django documentation,” 2005-2017. [Online]. Available: https://docs.djangoproject.com/en/1.10/.

[2] Django Project, «Documentación de Django,» 2005-2017. [En línea]. Available: https://docs.djangoproject.com/es/1.10/.

[3] Python, «Python 3.6.0 documentation,» 2001-2017. [En línea]. Available: https://docs.python.org/3/.

[4] S. Infante Montero, «Maestros del Web,» 7 mayo 2012. [En línea]. Available: http://www.maestrosdelweb.com/curso-django-el-modelo-de-datos/.

[5] The GnuCash Project, “Documentation,” 2001-2016. [Online]. Available: http://gnucash.org/docs.phtml.

[6] T. G. Project, “SQL - GnuCash,” [Online]. Available: https://wiki.gnucash.org/wiki/SQL.

[7] D. A. N. Harrington, “Hands-on Python Tutorial,” 18 enero 2014. [Online]. Available: http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html.

[8] CODEHERO, «Django desde Cero,» 2013-2015. [En línea]. Available: http://codehero.co/series/django-desde-cero.html.

[9] DjangoGirls, «Introducción · Django Girls Tutorial,» [En línea]. Available: https://tutorial.djangogirls.org/es/.

[10] Maestros del Web, «Categoría "django",» 2012. [En línea]. Available: http://www.maestrosdelweb.com/guias/guias-django/.

[11] R. M. Nikos Drakos, «Guía de aprendizaje de Python 5. Estructuras de datos,» 11 septiembre 2005. [En línea]. Available: http://pyspanishdoc.sourceforge.net/tut/node7.html.

[12] J. K.-M. Adrian Holovaty, El libro de Django 1.0, 2007.

[13] R. Gruet, “Python 2.7 Quick Reference,” 16 abril 2013. [Online]. Available: http://rgruet.free.fr/PQR27/PQR2.7.html.

[14] R. Brown, “Django vs Flask vs Pyramid: Choosing a Python Web Framework,” 2016. [Online]. Available: https://www.airpair.com/python/posts/django-flask-pyramid.

[15] R. Necaise, “Python for Java Programmers | Main / Table of Contents,” 2006-2008. [Online]. Available: http://python4java.necaiseweb.org/Main/TableOfContents.

[16] «Django-es - Grupos de Google,» [En línea]. Available: https://groups.google.com/forum/#!forum/django-es.

[17] A. Sarasa Cabezuelo, Gestión de la información web usando Python, Editorial UOC, 2016.

[18] Colaboradores de Wikipedia, «Django (framework),» Wikipedia, La enciclopedia libre, 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Django_(framework).

[19] Colaboradores de Wikipedia, «GnuCash,» Wikipedia, La enciclopedia libre, 2016. [En línea]. Available: https://es.wikipedia.org/wiki/GnuCash.

[20] Colaboradores de Wikipedia, «Ruby on Rails,» Wikipedia, La enciclopedia libre, 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Ruby_on_Rails.

[21] Colaboradores de Wikipedia, «PHP,» Wikipedia, La enciclopedia libre, 2017. [En línea]. Available: https://es.wikipedia.org/wiki/PHP.

[22] Colaboradores de Wikipedia, «Python,» Wikipedia, La enciclopedia libre, 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Python.

[23] Colaboradores de Wikipedia, «MySQL,» Wikipedia, La enciclopedia libre, 2017. [En línea]. Available: https://es.wikipedia.org/wiki/MySQL.

[24] W. Rosita, M. Margarita, C. Maximiliano, M. Marcos y P. Nicolás, «Algoritmos de Programación con Python,» [En línea]. Available: http://librosweb.es/libro/algoritmos_python/.

Glosario

ANSI C
: Es un estándar publicado por el Instituto Nacional Estadounidense de Estándares (ANSI), para el lenguaje de programación C., 8

API
: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción., 13

APT
: Advanced Packaging Tool (Herramienta Avanzada de Empaquetado), abreviado APT, es un sistema de gestión de paquetes creado por el proyecto Debian., 17, 19

BBDD
: Bases de datos, 7

C
: Es un lenguaje de programación, como evolución del anterior lenguaje B, a su vez basado en BCPL., 12

CouchDB
: Apache CouchDB, comúnmente llamada CouchDB, es un gestor de bases de datos de código abierto, cuyo foco está puesto en la facilidad de su uso y en ser "una base de datos que asume la web de manera completa"., 14

CRUD
: Create, Read, Update and Delete o "Crear, Leer, Actualizar y Borrar" en español. Se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software., 17

CSS
: Hojas de estilo en cascada (o CSS, siglas en inglés de Cascading Stylesheets) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado., 28, 40, 47

DOS
: Disk Operating System, "Sistema Operativo de Disco" y "Sistema Operativo en Disco", es una familia de sistemas operativos para computadoras personales (PC)., 14

framework
: Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software., 6, 13, 17, 19, 37, 42, 44

GET
Método de HTTP para recuperar datos de un servidor. El método GET solicita un recurso del servidor indicado en el campo URI., 31, 32, 36, 40

github
GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git., 20

GNU
Es un sistema operativo de tipo Unix desarrollado por y para el Proyecto GNU. Está formado en su totalidad por software libre, mayoritariamente bajo términos de copyleft., 2, 4, 9

HBCI
Homebanking Computer Interface. Archivo de almacenamiento de datos de cuentas., 10

HTML
HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web., 6, 7, 10, 12, 22, 26, 27, 28, 31, 32, 36, 37, 40, 42, 44

HTTPS
Hypertext Transfer Protocol Secure (Protocolo seguro de transferencia de hipertexto), es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP., 43, 44

IRC
Internet Relay Chat. Es un protocolo de comunicación en tiempo real basado en texto, que permite debates entre dos o más personas., 48

Java
Lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible., 8, 17

JavaScript
Lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,3 basado en prototipos, imperativo, débilmente tipado y dinámico. Utilizado en el lado del cliente., 12

Jinja2
Motor de templates para Python., 13

LAMP
LAMP es el acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas
Linux, Apache, MySQL/MariaDB, Perl/PHP/Python., 12

LAMPP
Paquete Linux para la instalación LAMP. Similar a XAMPP, 17, 19

licencia BSD
Berkeley Software Distribution. Es una licencia de software libre permisiva, 6, 13

Licencia GNU "GPL"
: General Public License. Es la licencia de derecho de autor más ampliamente usada en el mundo del software libre y código abierto, y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el software, 8, 9

OFX
: Open Financial Exchange File. Archivo de almacenamiento de datos de cuentas, 10

ORM
: Object-relationl Mapping, 17

POST
: Método de HTTP para recuperar datos de un servidor. El método POST se utiliza para pasar explícitamente datos al servidor en el propio mensaje de solicitud., 40

Proteccion "CSRF"
: Cross-site request forgery o falsificación de petición en sitios cruzados) es un tipo de exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía., 7

Pylons
: Es un framework web ligero y hace mucho énfasis en la flexibilidad y el rápido desarrollo. escrito en Python., 13

request
: Solicitud que se realiza a un servidor mediante protocolo HTTP. El servidor procesa la solicitud del cliente y devuelve una respuesta., 22, 26, 36

shell
: Término usado en informática para referirse a un intérprete de comandos, el cual consiste en la interfaz de usuario tradicional de los sistemas operativos basados en Unix y similares como GNU/Linux., 5

SO
: Sistema Operativo, 8

SQL
: Structured Query Language. En español lenguaje de consulta estructurada. Es un lenguaje específico del dominio que da acceso a un sistema de gestión de bases de datos relacionales que permite especificar diversos tipos de operaciones en ellos., 8, 12, 14, 18

SQLAlchemy
: Es un ORM para python. Incluye soporte para SQLite, MySQL, PostgreSQL, Oracle, MS SQL, entre otros., 14

SSH
: Secure SHell, en español
  intérprete de órdenes seguro. Es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red., 19

SVN
: Apache Subversion. Es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros., 48

URL
: Uniform Resource Locator. Es un identificador de recursos uniforme (Uniform Resource Identifier, URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo., IV, 7, 13, 14, 26, 31, 32, 43, 45

WML
: (Wireless Markup Language) es el lenguaje para crear páginas que puedan ser desplegadas en los móviles a través de interfases visuales de los micro-navegadores (WAP Browsers) hechos para tal fin. WML está definido como una aplicación XML 1.0., 12

WSGI
: Web Server Gateway Interface. WSGI es una interface simple y universal entre los servidores web y las aplicaciones web o frameworks., 13

XML
: eXtensible Markup Language o "Lenguaje de Marcas Extensible" es un meta-lenguaje que permite definir lenguajes de marcas, utilizado para almacenar datos de forma legible., 1, 3, 10, 12, 16

Zope
: Zope es un proyecto comunitario activista de un entorno de desarrollo para la creación de sitios web dinámicos y/o aplicaciones web usando un servidor de aplicaciones web orientado al objeto, escrito en el lenguaje de programación Python., 13, 14